httpstate	A. Barth	
Internet-Draft	U.C. Berkeley	
Obsoletes: 2109 (if approved)	April 17, 2010	
Intended status: Standards Track		
Expires: October 19, 2010		

TOC

HTTP State Management Mechanism
draft-ietf-httpstate-cookie-06

Abstract

This document defines the HTTP Cookie and Set-Cookie headers. These headers can be used by HTTP servers to store state on HTTP user agents, letting the servers maintain a stateful session over the mostly stateless HTTP protocol. The cookie protocol has many historical infelicities that degrade its security and privacy.

> NOTE: 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. Using the Set-Cookie header, an HTTP server can store name/value pairs and associated metadata (called cookies) at the user agent. When the user agent makes subsequent requests to the server, the user agent uses the metadata to determine whether to return the name/value pairs in the Cookie header.

Although simple on its surface, the cookie protocol has a number of complexities. For example, the server indicates a scope for each cookie when sending them to the user agent. The scope indicates the maximum amount of time the user agent should retain the cookie, to which servers the user agent should return the cookie, and for which protocols the cookie is applicable.

For historical reasons, the cookie protocol contains a number of security and privacy infelicities. For example, a server can indicate that a given cookie is intended for "secure" connections, but the Secure attribute provides only confidentiality (not integrity) from active network attackers. Similarly, cookies for a given host are shared across all the ports on that host, even though the usual "same-origin policy" used by web browsers isolates content retrieved from different ports.

TOC

2. General Nonsense

2.1. Conformance Criteria

TOC

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in document are to be interpreted as described in [RFC2119] (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.).
Requirements phrased in the imperative as part of algorithms (such as "strip any leading space characters" or "return false and abort these steps") are to be interpreted with the meaning of the key word ("MUST", "SHOULD", "MAY", etc) used in introducing the algorithm.

2.2. Syntax Notation

TOC

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] (Crocker, D., Ed. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” January 2008.).
The following core rules are included by reference, as defined in [RFC5234] (Crocker, D., Ed. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” January 2008.), Appendix B.1: ALPHA (letters), CR (carriage return), CRLF (CR LF), CTL (controls), DIGIT (decimal 0-9), DQUOTE (double quote), HEXDIG (hexadecimal 0-9/A-F/a-f), LF (line feed), OCTET (any 8-bit sequence of data), SP (space), HTAB (horizontal tab), VCHAR (any visible [USASCII] character), and WSP (whitespace).
The OWS (optional whitespace) rule is used where zero or more linear whitespace characters may appear. OWS SHOULD either not be produced or be produced as a single SP character. Multiple OWS characters that occur within field-content SHOULD be replaced with a single SP before interpreting the field value or forwarding the message downstream.

2.3. Terminology

TOC

The terms user agent, client, server, proxy, and origin server have the same meaning as in the HTTP/1.1 specification ([RFC2616] (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.)).
The terms request-host and request-URI refer to the values the user agent 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.

---

3. Overview

TOC

We outline here a way for an origin server to send state information to a user agent, and for the user agent to return the state information to the origin server.
To initiate a session, the origin server includes 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).
The user agent returns a Cookie request header to the origin server if it chooses to continue a session. The Cookie header contains a number of cookies the user agent received in previous Set-Cookie headers. The origin server MAY ignore the Cookie header or use the header to determine the current state of the session. The origin server MAY send the user agent 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 header with any response. User agents SHOULD send a Cookie request header, subject to other rules detailed below, with every request.
An origin server MAY include multiple Set-Cookie header fields in a single response. Note that an intervening gateway MUST NOT fold multiple Set-Cookie header fields into a single header field.
If a server sends multiple responses containing Set-Cookie headers concurrently to the user agent (e.g., when communicating with the user agent over multiple sockets), these responses create a "race condition" that can lead to unpredictable behavior.

3.1. Examples

TOC

Using the cookie protocol, a server can send the user agent a short string in an HTTP response that the user agent will return in future HTTP requests. For example, the server can send the user agent a "session identifier" named SID with the value 31d4d96e407aad42. The user agent then returns the session identifier in subsequent requests.

```
== Server -> User Agent ==
Set-Cookie: SID=31d4d96e407aad42

== User Agent -> Server ==
Cookie: SID=31d4d96e407aad42
```

The server can alter the default scope of the cookie using the Path and Domain attributes. For example, the server can instruct the user agent to return the cookie to every path and every subdomain of example.com.

```
== Server -> User Agent ==
Set-Cookie: SID=31d4d96e407aad42; Path=/; Domain=.example.com

== User Agent -> Server ==
Cookie: SID=31d4d96e407aad42
```

The server can store multiple cookies in the user agent. For example, the server can store a session identifier as well as the user's preferred language by returning two Set-Cookie response headers. Notice that the server uses the Secure and HttpOnly attributes to provide additional security protections for the more-sensitive session identifier.

```
== Server -> User Agent ==
Set-Cookie: SID=31d4d96e407aad42; Path=/; Secure; HttpOnly
Set-Cookie: lang=en-US; Path=/; Domain=.example.com

== User Agent -> Server ==
Cookie: SID=31d4d96e407aad42; lang=en-US
```

If the server wishes the user agent to persist the cookie over multiple sessions, the server can specify a expiration date in the Expires attribute. Note that the user agent might might delete the cookie before the expiration date if the user agent's cookie store exceeds its quota or if the user manually deletes the server's cookie.

```
== Server -> User Agent ==
Set-Cookie: lang=en-US; Expires=Wed, 09 Jun 2021 10:18:14 GMT

== User Agent -> Server ==
Cookie: lang=en-US
```

Finally, to remove a cookie, the server returns a Set-Cookie header with an expiration date in the past. The server will be successful in removing the cookie only if the Path and the Domain attribute in the Set-Cookie header match the values used when the cookie was created.

```
== Server -> User Agent ==
Set-Cookie: lang=; Expires=Sun, 06 Nov 1994 08:49:37 GMT

== User Agent -> Server ==
(No Cookie header)
```

---

4. A Well-Behaved Profile

TOC

This section describes the syntax and semantics of a well-behaved profile of the protocol. Servers SHOULD use the profile described in this section, both to maximize interoperability with existing user agents and because a future version of the cookie protocol could remove support for some of the most esoteric aspects of the protocol. User agents, however, MUST implement the full protocol to ensure interoperability with servers making use of the full protocol.

4.1. Set-Cookie

TOC

The Set-Cookie header is used to send cookies from the server to the user agent.

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 attribute-value pairs. Servers SHOULD NOT send Set-Cookie headers that fail to conform to the following grammar:

```
set-cookie-header = "Set-Cookie:" OWS set-cookie-string OWS
set-cookie-string = cookie-pair *( ";" cookie-av )
cookie-pair       = cookie-name "=" cookie-value
cookie-name       = token
cookie-value      = token
token             = <token, as defined in RFC 2616>

cookie-av         = expires-av / max-age-av / domain-av /
                    path-av / secure-av / httponly-av
expires-av        = "Expires" "=" sane-cookie-date
sane-cookie-date  = <rfc1123-date, as defined in RFC 2616>
max-age-av        = "Max-Age" "=" 1*DIGIT
domain-av         = "Domain" "=" domain-value
domain-value      = token
path-av           = "Path" "=" path-value
path-value        = <abs_path, as defined in RFC 2616>
secure-av         = "Secure"
httponly-av       = "HttpOnly"
```

Servers SHOULD NOT include two attributes with the same name.
Servers SHOULD NOT include two Set-Cookie header fields in the same response with the same cookie-name.
The cookie-value is opaque to the user agent and MAY be anything the origin server chooses to send. "Opaque" implies that the content is of interest and relevance only to the origin server. The content is, in fact, readable by anyone who examines the Set-Cookie header.
To maximize compatibility with user agents, servers that wish to store non-ASCII data in a cookie-value SHOULD encode that data using a printable ASCII encoding, such as base64.
NOTE: Some user agents represent dates using 32-bit integers. Some of these user agents might contain bugs that cause them process dates after the year 2038 incorrectly. Servers wishing to interoperate with these user agents might wish to use dates before 2038.
NOTE: The syntax above allows whitespace between the attribute and the U+003D ("=") character. Servers wishing to interoperate with some legacy user agents might wish to omit this whitespace.

4.1.2. Semantics (Non-Normative)

TOC

This section describes a simplified semantics of the Set-Cookie header. These semantics are detailed enough to be useful for understanding the most common uses of the cookie protocol. The full semantics are described in Section 5 (The Cookie Protocol).

When the user agent receives a Set-Cookie header, the user agent stores the cookie in its cookie store. When the user agent subsequently makes an HTTP request, the user agent consults its cookie store and includes the applicable, non-expired cookies in the Cookie header.
If the cookie store already contains a cookie with the same cookie-name, domain-value, and path-value, the existing cookie is evicted from the cookie store and replaced with the new value. Notice that servers can delete cookies by including an Expires attribute with a value in the past.
Unless the cookie's attributes indicate otherwise, the cookie is returned only to the origin server, and it expires at the end of the current session (as defined by the user agent). User agents ignore unrecognized cookie attributes.

4.1.2.1. Expires

TOC

The Expires attribute indicates the maximum lifetime of the cookie, represented as the date and time at which the cookie expires. The user agent is not required to retain the cookie until the specified date has passed. In fact, user agents often evict cookies from the cookie store due to memory pressure or privacy concerns.

4.1.2.2. Max-Age

TOC

The Max-Age attribute indicates the maximum lifetime of the cookie, represented as the number of seconds until the cookie expires. The user agent is not required to retain the cookie until the specified date has passed. In fact, user agents often evict cookies from the cookie store due to memory pressure or privacy concerns.

> WARNING: Not all user agents support the Max-Age attribute. User agents that do not support the Max-Age attribute will retain the cookie for the current session only.

If a cookie has both the Max-Age and the Expires attribute, the Max-Age attribute has precedence and controls the expiration date of the cookie.

TOC

4.1.2.3. Domain

The Domain attribute specifies those hosts for which the cookie will be sent. For example, if the Domain attribute contains the value ".example.com", the user agent will include the cookie in the Cookie header when making HTTP requests to example.com, www.example.com, and www.corp.example.com. (Note that a leading U+002E ("."), if present, is ignored.) If the server omits the Domain attribute, the user agent will return the cookie only to the origin server.

> WARNING: Some legacy user agents treat an absent Domain attribute as if the Domain attribute were present and contained the current host name. For example, if example.com returns a Set-Cookie header without a Domain attribute, these user agents will send the cookie to www.example.com.

The user agent will reject cookies (refuse to store them in the cookie store) unless the Domain attribute specifies a scope for the cookie that would include the origin server. For example, the user agent will accept a Domain attribute of ".example.com" or of ".foo.example.com" from foo.example.com, but the user agent will not accept a Domain attribute of ".bar.example.com" or of ".baz.foo.example.com".
NOTE: For security reasons, some user agents are configured to reject Domain attributes that do not correspond to a "registry controlled" domain (or a subdomain of a registry controlled domain). For example, some user agents will reject Domain attributes of ".com" or ".co.uk".

4.1.2.4. Path

TOC

The Path attribute limits the scope of the cookie to a set of paths. When a cookie has a Path attribute, the user agent will include the cookie in an HTTP request only if the path portion of the Request-URI matches (or is a subdirectory of) the cookie's Path attribute, where the U+002F ("/") character is interpreted as a directory separator. If the server omits the Path attribute, the user agent will use the directory of the Request-URI's path component as the default value. Although seemingly useful for isolating cookies between different paths within a given domain, the Path attribute cannot be relied upon for security for two reasons: First, user agents do not prevent one path from overwriting the cookies for another path. For example, if a response to a request for /foo/bar.html attempts to set a cookie with a Path attribute of "/baz" the user agent will store that cookie in the cookie store. Second, the "same-origin" policy implemented by many user agents does not isolate different paths within an origin. For example, /foo/bar.html can read cookies with a Path attribute of "/baz" because they are within the "same origin".

4.1.2.5. Secure

TOC

The Secure attribute limits the scope of the cookie to "secure" channels (where "secure" is defined by the user agent). When a cookie has the Secure attribute, the user agent will include the cookie in an HTTP request only if the request is transmitted over a secure channel (typically TLS [RFC5246] (Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” August 2008.)). Although seemingly useful for protecting cookies from active network attackers, the Secure attribute protects only the cookie's confidentiality. An active network attacker can overwrite Secure cookies from an insecure channel, disrupting the integrity of the cookies.

4.1.2.6. HttpOnly

TOC

The HttpOnly attribute limits the scope of the cookie to HTTP requests. In particular, the attribute instructs the user agent to omit the cookie when providing access to its cookie store via "non-HTTP" APIs (as defined by the user agent).

4.2. Cookie

TOC

4.2.1. Syntax

TOC

The user agent returns stored cookies to the origin server in the Cookie header. If the server conforms to the requirements in this section, the requirements in the next section will cause the user agent to return a Cookie header that conforms to the following grammar:

```
cookie-header = "Cookie:" OWS cookie-string OWS
cookie-string = cookie-pair *( ";" cookie-pair )
```

TOC

4.2.2. Semantics

Each cookie-pair represents a cookie stored by the user agent. The cookie-name and the cookie-value are returned verbatim from the corresponding parts of the Set-Cookie header.
Notice that the cookie attributes are not returned. In particular, the server cannot determine from the Cookie header alone when a cookie will expire, for which domains the cookie is valid, for which paths the cookie is valid, or whether the cookie was set with the Secure or HttpOnly attributes.
The semantics of individual cookies in the Cookie header is not defined by this document. Servers are expected to imbue these cookies with server-specific semantics.
Although cookies are serialized linearly in the Cookie header, servers SHOULD NOT rely upon the serialization order. In particular, if the Cookie header contains two cookies with the same name, servers SHOULD NOT rely upon the order in which these cookies appear in the header.

---

5. The Cookie Protocol

TOC

For historical reasons, the full cookie protocol contains a number of exotic quirks. This section is intended to specify the cookie protocol in enough detail to enable a user agent that implements the protocol precisely as specified to interoperate with existing servers.
Conformance requirements phrased as algorithms or specific steps may be implemented in any manner, so long as the end result is equivalent. (In particular, the algorithms defined in this specification are intended to be easy to follow, and not intended to be performant.)

5.1. Algorithms

TOC

This section defines a number of algorithms used by the cookie protocol.

5.1.1. Dates

TOC

The user agent MUST use the following algorithm to *parse a cookie-date*:

1. Using the grammar below, divide the cookie-date into date-tokens.

```
cookie-date     = *delimiter date-token-list *delimiter
date-token-list = date-token *( 1*delimiter date-token )
delimiter       = %x09 / %x20 / %x21 / %x22 / %x23 / %x24 /
                  %x25 / %x26 / %x27 / %x28 / %x29 / %x2A /
                  %x2B / %x2C / %x2D / %x2E / %x2F / %x3B /
                  %x3C / %x3D / %x3E / %x3F / %x40 / %x5B /
                  %x5C / %x5D / %x5E / %x5F / %x60 / %x7B /
                  %x7C / %x7D / %x7E
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            = time-field ":" time-field ":" time-field
time-field      = 2DIGIT / DIGIT
mystery         = <anything except a delimiter>
```

2. Process each date-token sequentially in the order the date-tokens appear in the cookie-date:

    1. If the found-day-of-month flag is not set and the date-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 date-token. Skip the remaining sub-steps and continue to the next date-token.

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

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

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

3. Abort these steps and *fail to parse* 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.

4. If the year-value is greater than 68 and less than 100, increment the year-value by 1900.

5. If the year-value is greater than or equal to 0 and less than 69, increment the year-value by 2000.

6. Let the parsed-cookie-date 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.

7. Return the parsed-cookie-date as the result of this algorithm.

5.1.2. Domains

TOC

A *canonicalized* host-name is the host-name converted to lower case and expressed in punycode [RFC3492] (Costello, A., “Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA),” March 2003.).
A request-host *domain-matches* a cookie-domain if at least one of the following conditions hold:

*The cookie-domain and the canonicalized request-host are identical.

*All of the following conditions hold:

-The cookie-domain is a suffix of the canonicalized request-host.

-The last character of the canonicalized request-host that is not included in the cookie-domain is a U+002E (".") character.

-The request-host is a host name (i.e., not an IP address).

5.1.3. Paths

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 Request-URI.

2. If the first character of the uri-path is not a U+002F ("/") character, output U+002F ("/") and skip the remaining steps.

3. If the uri-path contains only a single U+002F ("/") character, output U+002F ("/") and skip the remaining steps.

4. Output the characters of the uri-path from the first character up to, but not including, the right-most U+002F ("/").

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

*The cookie-path and the request-path are identical.

*The cookie-path is a prefix of the request-path and the last character of the cookie-path is U+002F ("/").

*The cookie-path is a prefix of the request-path and the first character of the request-path that is not included in the cookie-path is a U+002F ("/") character.

5.2. The Set-Cookie Header

TOC

When a user agent receives a 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. If the set-cookie-string is empty or consists entirely of WSP characters, the user agent MAY ignore the set-cookie-string entirely.

2. If the set-cookie-string contains a U+003B (";") character:

   > The name-value-pair string consists of the characters up to, but not including, the first U+003B (";"), and the unparsed-attributes consist of the remainder of the set-cookie-string (including the U+003B (";") in question).

   Otherwise:

   > The name-value-pair string consists of all the characters contained in the set-cookie-string, and the unparsed-attributes is the empty string.

3. If the name-value-pair string lacks a U+003D ("=") character, ignore the set-cookie-string entirely.

4. If the first character of the name-value-pair string is U+003D ("="), ignore the set-cookie-string entirely.

5. The (necessarily non-empty) name string consists of the characters up to, but not including, the first U+003D ("=") character, and the (possibly empty) value string consists of the characters after the first U+003D ("=") character.

6. Remove any leading or trailing WSP characters from the name string and the value string.

7. 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+003B (";") character).

3. If the remaining unparsed-attributes contains a U+003B (";") character:

> Consume the characters of the unparsed-attributes up to, but not including, the first U+003B (";") character.

Otherwise:

> Consume the remainder of the unparsed-attributes.

Let the cookie-av string be the characters consumed in this step.

4. If the cookie-av string contains a U+003D ("=") character:

   > The (possibly empty) attribute-name string consists of the characters up to, but not including, the first U+003D ("=") character, and the (possibly empty) attribute-value string consists of the characters after the first U+003D ("=") character.

   Otherwise:

   > The attribute-name string consists of the entire cookie-av string, and the attribute-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 WSP characters from the attribute-name string and the attribute-value string.

6. Process the attribute-name and attribute-value according to the requirements in the following subsections.

7. Return to Step 1.

When the user agent finishes parsing the set-cookie-string, the user agent *receives a cookie* from the Request-URI with name cookie-name, value cookie-value, and attributes cookie-attribute-list.

5.2.1. The Max-Age Attribute

TOC

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

If delta-seconds is less than or equal to zero (0), let expiry-time be the current date and time. Otherwise, let the expiry-time be the current date and time plus delta-seconds seconds.
Append an attribute to the cookie-attribute-list with an attribute-name of Max-Age and an attribute-value of expiry-time.

5.2.2. The Expires Attribute

TOC

If the attribute-name case-insensitively matches the string "Expires", the user agent MUST process the cookie-av as follows.
Let the parsed-cookie-date be the result of parsing the attribute-value as cookie-date.
If the attribute-value failed to parse as a cookie date, ignore the cookie-av.
If the user agent received the set-cookie-string from an HTTP response that contains a Date header field and the contents of the last Date header field successfully parse as a cookie-date:

> Let server-date be the date obtained by parsing the contents of the last Date header field as a cookie-date.
>
> Let delta-seconds be the number of seconds between the server-date and the parsed-cookie-date (i.e., parsed-cookie-date - server-date).
>
> Let the expiry-time be the current date and time plus delta-seconds seconds.

Otherwise:

> Let the expiry-time be the parsed-cookie-date.

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 to the cookie-attribute-list with an attribute-name of Expires and an attribute-value of expiry-time.

5.2.3. The Domain Attribute

TOC

If the attribute-name case-insensitively matches the string "Domain", the user agent MUST process the cookie-av as follows.

If the attribute-value is empty, the behavior is undefined. However, user agent SHOULD ignore the cookie-av entirely.
If the first character of the attribute-value string is U+002E ("."):

> Let cookie-domain be the attribute-value without the leading U+002E (".") character.

Otherwise:

> Let cookie-domain be the entire attribute-value.

Convert the cookie-domain to lower case.
Append an attribute to the cookie-attribute-list with an attribute-name of Domain and an attribute-value of cookie-domain.

5.2.4. The Path Attribute

TOC

If the attribute-name case-insensitively matches the string "Path", the user agent MUST process the cookie-av as follows.
If the attribute-value is empty or if the first character of the attribute-value is not U+002F ("/"):

> Let cookie-path be the default-path.

Otherwise:

> Let cookie-path be the attribute-value.

Append an attribute to the cookie-attribute-list with an attribute-name of Path and an attribute-value of cookie-path.

5.2.5. The Secure Attribute

TOC

If the attribute-name case-insensitively matches the string "Secure", the user agent MUST append an attribute to the cookie-attribute-list with an attribute-name of Secure and an empty attribute-value.

5.2.6. The HttpOnly Attribute

TOC

If the attribute-name case-insensitively matches the string "HttpOnly", the user agent MUST append an attribute to the cookie-attribute-list with an attribute-name of HttpOnly and an empty attribute-value.

5.3. Storage Model

TOC

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 a received cookie in its entirety if the user agent is configured to block receiving cookies. 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, value, expiry-time, domain, path, creation-time, last-access-time, persistent-flag, host-only-flag, secure-only-flag, and http-only-flag.
When the user agent receives a cookie from a Request-URI with name cookie-name, value cookie-value, and attributes cookie-attribute-list, the user agent MUST process the cookie as follows:

1. Create a new cookie with name cookie-name, value cookie-value. Set the creation-time and the last-access-time to the current date and time.

2. If the cookie-attribute-list contains an attribute with an attribute-name of "Max-Age":

   > Set the cookie's persistent-flag to true.
   >
   > Set the cookie's expiry-time to attribute-value of the last attribute in the cookie-attribute-list with an attribute-name of "Max-Age".

   Otherwise, if the cookie-attribute-list contains an attribute with an attribute-name of "Expires" (and does not contain an attribute with an attribute-name of "Max-Age"):

   > Set the cookie's persistent-flag to true.
   >
   > Set the cookie's expiry-time to attribute-value of the last attribute in the cookie-attribute-list with an attribute-name of "Expires".

   Otherwise:

   > Set the cookie's persistent-flag to false.
   >
   > Set the cookie's expiry-time to the latest representable date.

3. If the cookie-attribute-list contains an attribute with an attribute-name of "Domain":

   > Let the domain-attribute be the attribute-value of the last attribute in the cookie-attribute-list with an attribute-name of "Domain".

   Otherwise:

   > Let the domain-attribute be the empty string.

4. If the user agent is configured to use a "public suffix" list and the domain-attribute is a public suffix:

   > If the domain-attribute is identical to the canonicalized Request-URI's host:
   >
   > > Let the domain-attribute be the empty string.
   >
   > Otherwise:
   >
   > > Ignore the cookie entirely and abort these steps
   >
   > NOTE: A "public suffix" is a domain that is controlled by a public registry, such as "com", "co.uk", and "pvt.k12.wy.us". This step is essential for preventing attacker.com from disrupting the integrity of example.com by setting a cookie with a Domain attribute of "com". Unfortunately, the set of public suffixes (also known as "registry controlled domains") changes over time. If feasible, user agents SHOULD use an up-to-date public suffix list, such as the one maintained by the Mozilla project at http://publicsuffix.org/.

5. If the domain-attribute is non-empty:

   > If the Request-URI's host does not domain-match the domain-attribute, ignore the cookie entirely and abort these steps.
   >
   > Set the cookie's host-only-flag to false.
   >
   > Set the cookie's domain to the domain-attribute.

   Otherwise:

   > Set the cookie's host-only-flag to true.
   >
   > Set the cookie's domain to the host of the Request-URI.

6. If the cookie-attribute-list contains an attribute with an attribute-name of "Path", set the cookie's path to attribute-value of the last attribute in the cookie-attribute-list with an attribute-name of "Path". Otherwise, set cookie's path to the default-path of the Request-URI.

7. If the cookie-attribute-list contains an attribute with an attribute-name of "Secure", set the cookie's secure-only-flag to true. Otherwise, set cookie's secure-only-flag to false.

8. If the cookie-attribute-list contains an attribute with an attribute-name of "HttpOnly", set the cookie's http-only-flag to true. Otherwise, set cookie's http-only-flag to false.

9. If the cookie's name and value are both empty, abort these steps and ignore the cookie entirely.

10. If the cookie's expiry-time is not in the future, abort these steps and ignore the cookie entirely.

11. If the cookie was received from a non-HTTP context and the cookie's http-only-flag is set, abort these steps and ignore the cookie entirely.

12. If the cookie store contains a cookie with the same name, domain, and path as the newly created cookie:

    1. Let old-cookie be the existing cookie with the same name, domain, and path as the newly created cookie. (Notice that this algorithm maintains the invariant that there is at most one such cookie.)

    2. If the newly created cookie was received from an non-HTTP context and the old-cookie's host-only-flag is set, abort these steps and ignore the newly created cookie entirely.

    3. Update the creation-time of the newly created cookie to match the creation-time of the old-cookie.

    4. Remove the old-cookie from the cookie store.

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

The user agent MUST evict a cookie from the cookie store if, at any time, 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).

The user agent MAY evict a cookie from the cookie store if the cookie store exceeds some predetermined upper bound (such as 3000 cookies). When the user agent evicts a cookie 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 with more than a predetermined number of other cookies.

3. All 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" (as defined by the user agent), the user agent MUST remove from the cookie store all cookies with the persistent-flag set to false.

5.4. The Cookie Header

TOC

When the user agent generates an HTTP request, the user agent SHOULD attach exactly one HTTP header named Cookie if the cookie-string (defined below) for the Request-URI is non-empty.
A user agent MAY omit the Cookie header in its entirety if the user agent is configured to block sending cookies. 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 a Request-URI:

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

   *Let request-host be the Request-URI's host. Either:

   > The cookie's host-only-flag is true and the canonicalized request-host is identical to the cookie's domain.

   Or:

   > The cookie's host-only-flag is false and the request-host domain-matches cookie's domain.

   *The Request-URI's path patch-matches cookie's path.

*If the cookie's secure-only-flag is true, then the Request-URI's scheme must denote a "secure" protocol (as defined by the user agent).

> NOTE: The notion of a "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-flag is true, then exclude the cookie unless the cookie-string is being generated for an "HTTP" API (as defined by the user agent).

2. The user agent SHOULD sort the cookie-list in the following order:

   *Cookies with longer paths are listed before cookies with shorter paths.

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

   > NOTE: Not all user agents sort the cookie-list in this order, but this order reflects common practice when this document was written. The specific ordering might not be optimal in every metric, but using the consensus ordering is a relatively low cost way to improve interoperability between user agents.

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

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

   1. Output the cookie's name, the U+003D ("=") character, and the cookie's value.

   2. If there is an unprocessed cookie in the cookie-list, output the characters U+003B and U+0020 ("; ").

Note: Despite its name, the cookie-string is actually a sequence of octets, not a sequence of characters. To convert the cookie-string into a sequence of characters (e.g., for presentation to the user), the user agent SHOULD use the UTF-8 character encoding [RFC3629] (Yergeau, F., “UTF-8, a transformation format of ISO 10646,” November 2003.).

6. Implementation Considerations

TOC

6.1. 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 sum of the length of the cookie's name, value, and attributes).
- At least 50 cookies per domain.
- At least 3000 cookies total.

Servers SHOULD use as few and as small cookies as possible to avoid reaching these implementation limits and to avoid network latency due to the Cookie header being included in every request.
Servers should gracefully degrade if the user agent fails to return one or more cookies in the Cookie header because the user agent might evict any cookie at any time on orders from the user.

6.2. Application Programmer Interfaces

TOC

One reason the cookie protocol uses such an esoteric syntax is because many platforms (both in servers and user agents) provide string-based application programmer interfaces (APIs), requiring application-layer programmers to generate and parse the syntax used by the cookie protocol.
Instead of providing string-based APIs to the cookie protocols, implementations would be well-served by providing more semantic APIs. It is beyond the scope of this document to recommend specific API designs, but there are clear benefits to accepting a abstract "Date" object instead of a serialized date string.

7. Privacy Considerations

TOC

The cookie protocol is often criticized for letting servers track users. For example, a number of "web analytics" companies use cookies

to recognize when a user returns to a web site or visits another web site. Although cookies are not the only mechanism servers can use to track users across HTTP requests, cookies facilitate tracking because they are persistent across user agent sessions and can be shared between host names.

7.1. Third-Party Cookies

TOC

Particularly worrisome are so-called "third-party" cookies. In rendering an HTML document, a user agent often requests resources from other servers (such as advertising networks). These third-party servers can use the cookie protocol to track the user even if the user never visits the server directly.

Some user agents restrict how third-party cookies behave. For example, some user agents refuse to send the Cookie header in third-party requests. Other user agents refuse to process the Set-Cookie header in responses to third-party requests. Among user agents, there is a wide variety of third-party cookie policies. This document grants user agents wide latitude to experiment with third-party cookie policies that balance the privacy and compatibility needs of their users. However, this document does not endorse any particular third-party cookie policy.

Third-party cookie blocking policies are often ineffective at achieving their privacy goals if servers attempt to work around their restrictions to track users. In particular, two collaborating servers can often track users without using cookies at all.

7.2. User Controls

TOC

User agents SHOULD provide users with a mechanism for managing the cookies stored in the cookie store. For example, a user agent might let users delete all cookies received during a specified time period or all the cookies related to a particular domain. In addition, many user agent include a user interface element that lets users examine the cookies stored in the cookie store.

User agents SHOULD provide users with a mechanism for disabling cookies. When cookies are disabled, the user agent MUST NOT include a Cookie header in outbound HTTP requests and the user agent MUST NOT process Set-Cookie headers in inbound HTTP responses.

Some user agents provide users the option of preventing persistent storage of cookies across sessions. When configured thusly, user agents MUST treat all received cookies as if the persistent-flag were set to false.

Some user agents provide users with the ability to approve individual writes to the cookie store. In many common usage scenarios, these controls generate a large number of prompts. However, some privacy-conscious users find these controls useful nonetheless.

8. Security Considerations

TOC

8.1. Overview

TOC

The cookie protocol has a number of security and privacy pitfalls. In particular, cookies encourage developers to rely on ambient authority for authentication, often creating vulnerabilities such as cross-site request forgery. When storing session identifiers in cookies, developers often create session fixation vulnerabilities. Transport-layer encryption, such as that employed in HTTPS, is insufficient to prevent a network attacker from obtaining or altering a victim's cookies because the cookie protocol itself has various vulnerabilities (see "Weak Confidentiality" and "Weak Integrity", below). In addition, by default, the cookie protocol does not provide confidentiality or integrity from network attackers, even when used in conjunction with HTTPS.

8.2. Ambient Authority

TOC

A server that uses cookies to authenticate users can suffer security vulnerabilities because some user agents let remote parties issue HTTP requests from the user agent (e.g., via HTTP redirects and HTML forms). When issuing those requests, user agent attaches cookies even if the entity does not know the contents of the cookies, possibly letting the remote entity exercise authority at an unwary server.

Although this security concern goes by a number of names (e.g., cross-site request forgery, confused deputy), the issue stems from cookies being a form of ambient authority. Cookies encourage server operators to separate designation (in the form of URLs) from authorization (in the form of cookies). Consequently, the user agent might supply the authorization for a resource designated by the attacker, possibly causing the server or its clients to undertake actions designated by the attacker as though they were authorized by the user.

Instead of using cookies for authorization, server operators might wish to consider entangling designation and authorization by treating URLs

as capabilities. Instead of storing secrets in cookies, this approach stores secrets in URLs, requiring the remote entity to supply the secret itself. Although this approach is not a panacea, judicious use of these principles can lead to more robust security.

8.3. Clear Text

TOC

Unless sent over a secure channel (such as TLS), the information in the Set-Cookie and Cookie headers is transmitted in the clear.

1. All sensitive information conveyed in these 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.

Servers SHOULD encrypt and sign the contents of cookies when transmitting them to the user agent (even when sending the cookies over a secure channel). However, encrypting and signing cookie contents does not prevent an attacker from transplanting a cookie from one user agent to another or from replaying the cookie at a later time.
In addition to encrypting and signing the contents of every cookie, servers that require a higher level of security SHOULD use the cookie protocol only over a secure channel. When using the cookie protocol over a secure channel, servers SHOULD set the Secure attribute in every cookie. If a server does not set the Secure attribute, the protection provided by the secure channel will be largely moot.

8.4. Session Identifiers

TOC

Instead of storing session information directly in a cookie (where it might be exposed to or replayed by an attacker), servers commonly store a nonce (or "session identifier") in a cookie. When the server receives an HTTP request with a nonce, the server can look up state information associated with the cookie using the nonce as a key.
Using session identifier cookies limits the damage an attacker can cause if the attacker learns the contents of a cookie because the nonce is useful only for interacting with the server (unlike non-nonce cookie content, which might itself be sensitive). Furthermore, using a single nonce prevents an attacker from "splicing" together cookie content from

two interactions with the server, which could cause the server to behave unexpectedly.

Using session identifiers is not without risk. For example, the server SHOULD take care to avoid "session fixation" vulnerabilities. A session fixation attack proceeds in three steps. First, the attacker transplants a session identifier from his or her user agent to the victim's user agent. Second, the victim uses that session identifier to interact with the server, possibly imbuing the session identifier with the user's credentials or confidential information. Third, the attacker uses the session identifier to interact with server directly, possibly obtaining the user's authority or confidential information.

8.5. Weak Confidentiality

TOC

Cookies do not provide isolation by port. If a cookie is readable by a service running on one port, the cookie is also readable by a service running on another port of the same server. If a cookie is writable by a service on one port, the cookie is also writable by a service running on another port of the same server. For this reason, servers SHOULD NOT both run mutually distrusting services on different ports of the same host and use cookies to store security-sensitive information.

Cookies do not provide isolation by scheme. Although most commonly used with the http and https schemes, the cookies for a given host might also available to other schemes, such as ftp and gopher. Although this lack of isolation by scheme is most apparent in via non-HTTP APIs that permit access to cookies (e.g., HTML's document.cookie API), the lack of isolation by scheme is actually present in the cookie protocol itself (e.g., consider retrieving a URI with the gopher scheme via HTTP).

Cookies do not always provide isolation by path. Although the network-level protocol does not send cookie stored for one path to another, some user agents expose cookies via non-HTTP APIs, such as HTML's document.cookie API. Because some of these user agents (e.g., web browsers) do not isolate resources received from different paths, a resource retrieved from one path might be able to access cookies stored for another path.

8.6. Weak Integrity

TOC

Cookies do not provide integrity guarantees for sibling domains (and their subdomains). For example, consider foo.example.com and bar.example.com. The foo.example.com server can set a cookie with a Domain attribute of ".example.com" (possibly overwriting an existing ".example.com" cookie set by bar.example.com), and the user agent will

include that cookie in HTTP requests to bar.example.com. In the worst case, bar.example.com will be unable to distinguish this cookie from a cookie it set itself. The foo.example.com server might be able to leverage this ability to mount an attack against bar.example.com.
Even though the cookie protocol supports the Path attribute, the Path attribute does not provide any integrity protection because the user agent will accept an arbitrary Path attribute in a Set-Cookie header. For example, an HTTP response to a request for http://example.com/foo/bar can set a cookie with a Path attribute of "/qux". Consequently, servers SHOULD NOT both run mutually distrusting services on different paths of the same host and use cookies store security sensitive information.
An active network attacker can also inject cookies into the Cookie header sent to https://example.com/ by impersonating a response from http://example.com/ and injecting a Set-Cookie header. The HTTPS server at example.com will be unable to distinguish these cookies from cookies that it set itself in an HTTPS response. An active network attacker might be able to leverage this ability to mount an attack against example.com even if example.com uses HTTPS exclusively.
Servers can partially mitigate these attacks by encrypting and signing the contents of their cookies. However, using cryptography does not mitigate the issue completely because an attacker can replay a cookie he or she received from the authentic example.com server in the user's session, with unpredictable results.
Finally, an attacker might be able to force the user agent to delete cookies by storing large number of cookies. Once the user agent reaches its storage limit, the user agent will be forced to evict some cookies. Servers SHOULD NOT rely upon user agents retaining cookies.

8.7. Reliance on DNS

TOC

The cookie protocol relies upon the Domain Name System (DNS) for security. If the DNS is partially or fully compromised, the cookie protocol might fail to provide the security properties required by applications.

---

9. References

TOC

9.1. Normative References

TOC

[RFC2119]

	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).
[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.
[RFC3492]	Costello, A., "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)," RFC 3492, March 2003 (TXT).
[RFC3629]	Yergeau, F., "UTF-8, a transformation format of ISO 10646," STD 63, RFC 3629, November 2003 (TXT).
[RFC5234]	Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF," STD 68, RFC 5234, January 2008.
[RFC5246]	Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, August 2008.

9.2. Informative References

TOC

[RFC2109]	Kristol, D. and L. Montulli, "HTTP State Management Mechanism," RFC 2109, February 1997.

Appendix A. Acknowledgements

TOC

This document borrows heavily from RFC 2109 [RFC2109] (Kristol, D. and L. Montulli, "HTTP State Management Mechanism," February 1997.). We are indebted to David M. Kristol and Lou Montulli for their efforts to specify the cookie protocol. David M. Kristol, in particular, provided invaluable advice on navigating the IETF process. We would also like to thank Thomas Broyer, Tyler Close, Bil Corry, corvid, Roy T. Fielding, Blake Frantz, Eran Hammer-Lahav, Jeff Hodges, Achim Hoffmann, Georg Koppen, Dean McNamee, Mark Miller, Yngve N. Pettersen, Julian Reschke, Mark Seaborn, Maciej Stachowiak, Daniel Stenberg, David Wagner, Dan Winship, and Dan Witte for their valuable feedback on this document.

Author's Address

TOC

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