| | | |
|---|---|---|
| httpstate | A. Barth | |
| Internet-Draft | U.C. Berkeley | |
| Obsoletes: 2109 (if approved) | January 22, 2010 | |
| Intended status: Standards Track | | |
| Expires: July 26, 2010 | | |

TOC

**HTTP State Management Mechanism**
**draft-ietf-httpstate-cookie-02**

**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 and should be avoided for new applications of HTTP.

> 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.

---

### 1.1. 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).

---

## 2. Terminology

TOC

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

---

### 3.1. Examples

TOC

[TODO: Put some examples here.

## 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 / domain-av / path-av /
                    secure-av / httponly-av
expires-av        = "Expires" "=" cookie-date
cookie-date       = <rfc1123-date, as defined in RFC 2616>
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.
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 is, in fact, readable by anyone who examines the Set-Cookie header.
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 elide 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. Domain

TOC

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.
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".

---

#### 4.1.2.3. 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".

---

#### 4.1.2.4. 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 [RFC5234] (Crocker, D., Ed. and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” January 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.

---

#### 4.1.2.5. HttpOnly

TOC

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

---

### 4.2. Cookie

TOC

---

TOC

### 4.2.1. Syntax

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 )
cookie-pair   = cookie-name "=" cookie-value
cookie-name   = token
cookie-value  = token
token         = <token, as defined in Section 2.2 of RFC 2616>
```

---

### 4.2.2. Semantics

TOC

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.

---

## 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            = 2DIGIT ":" 2DIGIT ":" 2DIGIT
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. 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.

*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, and request-host is a host name (i.e., not an IP address). [TODO: Is this the right way to spec this???]

---

### 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: [TODO: This isn't exactly what IE or Firefox does.]

*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 contains a U+003D ("=") character:

   > The (possibly 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.

   Otherwise:

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

4. Remove any leading or trailing WSP characters 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+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 Expires (note the name conversion) 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.
[TODO: Test ".127.0.0.1" and "127.0.0.1"]
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. [TODO: We need more tests for this, including with " characters and with multiple Path attributes.]

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 Secure 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 "Expires":

   > 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". [TODO: Test that this really works when mixing Max-Age and 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".
>
> If the Request-URI's host does not domain-match the domain-attribute, ignore the cookie entirely and abort these steps.
>
> If the user agent is configured to use a "public suffix" list and the domain-attribute is a public suffix, 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/.
>
> 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.

4. 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.

5. 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.

6. 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.

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

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

9. If the cookie's expiry-time is not in the future, abort these steps.

10. 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 elide 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 field 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 field is true, then exclude the cookie unless the cookie-string is being generated for an "HTTP" API (as defined by the user agent).

2. 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.

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. If the cookie's name is non-empty, output the cookie's name followed by the U+003D ("=") character.

2. Output the cookie's value.

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

---

## 6. 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 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.

---

## 7. Security Considerations

TOC

---

### 7.1. Clear Text

TOC

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 their cookies. However, encrypting and signing cookies does not prevent an attacker from transplanting a cookie from one user agent to another.
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.

---

### 7.2. 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 machine 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 are also available to other schemes, such as ftp and gopher. This lack of isolation is most easily seen when a user agent retrieves a URI with a gopher scheme via HTTP, but the lack of isolation by scheme is also apparent via non-HTTP APIs that permit access to cookies, such as HTML's document.cookie API.

---

### 7.3. 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", 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.
Similarly, an active network attacker can 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 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.

---

## 8. Normative References

TOC


| | |
|---|---|
| [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. |
| [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. |


---

## Appendix A. Acknowledgements

TOC

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

---

## Author's Address

TOC


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