HTTP
Internet-Draft
Obsoletes: 6265 (if approved)
Intended status: Standards Track
Expires: 18 September 2025

S. Bingler, Ed.
M. West, Ed.
Google LLC
J. Wilander, Ed.
Apple, Inc

17 March 2025

Cookies: HTTP State Management Mechanism
draft-ietf-httpbis-rfc6265bis-20

Abstract

This document defines the HTTP Cookie and Set-Cookie header fields. These header fields can be used by HTTP servers to store state (called cookies) at HTTP user agents, letting the servers maintain a stateful session over the mostly stateless HTTP protocol. Although cookies have many historical infelicities that degrade their security and privacy, the Cookie and Set-Cookie header fields are widely used on the Internet. This document obsoletes RFC 6265.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-httpbis-rfc6265bis/.

Discussion of this document takes place on the HTTP Working Group mailing list (mailto:ietf-http-wg@w3.org), which is archived at https://lists.w3.org/Archives/Public/ietf-http-wg/. Working Group information can be found at https://httpwg.org/.

Source for this draft and an issue tracker can be found at https://github.com/httpwg/http-extensions/labels/6265bis.

Status of This Memo

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

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

Copyright Notice

Table of Contents

1. Introduction

This document defines the HTTP Cookie and Set-Cookie header fields. Using the Set-Cookie header field, an HTTP server can pass name/value pairs and associated metadata (called cookies) to a user agent. When the user agent makes subsequent requests to the server, the user agent uses the metadata and other information to determine whether to return the name/value pairs in the Cookie header field.

Although simple on their surface, cookies have a number of complexities. For example, the server indicates a scope for each cookie when sending it to the user agent. The scope indicates the maximum amount of time in which the user agent should return the cookie, the servers to which the user agent should return the cookie, and the connection types for which the cookie is applicable.

For historical reasons, cookies contain 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 does not provide integrity in the presence of an active network attacker. 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 via different ports.

This specification applies to developers of both cookie-producing servers and cookie-consuming user agents. Section 3.2 helps to clarify the intended target audience for each implementation type.

To maximize interoperability with user agents, servers SHOULD limit themselves to the well-behaved profile defined in Section 4 when generating cookies.

User agents MUST implement the more liberal processing rules defined in Section 5, in order to maximize interoperability with existing servers that do not conform to the well-behaved profile defined in Section 4.

This document specifies the syntax and semantics of these header fields as they are actually used on the Internet. In particular, this document does not create new syntax or semantics beyond those in use today. The recommendations for cookie generation provided in Section 4 represent a preferred subset of current server behavior, and even the more liberal cookie processing algorithm provided in Section 5 does not recommend all of the syntactic and semantic variations in use today. Where some existing software differs from the recommended protocol in significant ways, the document contains a note explaining the difference.

This document obsoletes [RFC6265].

2. Conventions

2.1. Conformance Criteria

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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.

Conformance requirements phrased as algorithms or specific steps can 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 understand and are not intended to be performant.

2.2. Syntax Notation

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234].

The following core rules are included by reference, as defined in [RFC5234], Appendix B.1: ALPHA (letters), CR (carriage return), CRLF (CR LF), CTLs (controls), DIGIT (decimal 0-9), DQUOTE (double quote), HEXDIG (hexadecimal 0-9/A-F/a-f), LF (line feed), NUL (null octet), OCTET (any 8-bit sequence of data except NUL), SP (space), HTAB (horizontal tab), CHAR (any [USASCII] character), VCHAR (any visible [USASCII] character), and WSP (whitespace).

The OWS (optional whitespace) and BWS (bad whitespace) rules are defined in Section 5.6.3 of [HTTP].

2.3. Terminology

The terms "user agent", "client", "server", "proxy", and "origin server" have the same meaning as in the HTTP/1.1 specification ([HTTP], Section 3).

The request-host is the name of the host, as known by the user agent, to which the user agent is sending an HTTP request or from which it is receiving an HTTP response (i.e., the name of the host to which it sent the corresponding HTTP request).

The term request-uri refers to "target URI" as defined in Section 7.1 of [HTTP].

Two sequences of octets are said to case-insensitively match each other if and only if they are equivalent under the i;ascii-casemap collation defined in [RFC4790].

The term string means a sequence of non-NUL octets.

The terms "active browsing context", "active document", "ancestor navigables", "container document", "content navigable", "dedicated worker", "Document", "inclusive ancestor navigables", "navigable", "opaque origin", "sandboxed origin browsing context flag", "shared worker", "the worker’s Documents", "top-level traversable", and "WorkerGlobalScope" are defined in [HTML].

"Service Workers" are defined in the Service Workers specification [SERVICE-WORKERS].

The term "origin", the mechanism of deriving an origin from a URI, and the "the same" matching algorithm for origins are defined in [RFC6454].

"Safe" HTTP methods include GET, HEAD, OPTIONS, and TRACE, as defined in Section 9.2.1 of [HTTP].

A domain's "public suffix" is the portion of a domain that is controlled by a public registry, such as "com", "co.uk", and "pvt.k12.wy.us". A domain's "registrable domain" is the domain's public suffix plus the label to its left. That is, for https://www.site.example, the public suffix is example, and the registrable domain is site.example. Whenever possible, user agents SHOULD use an up-to-date public suffix list, such as the one maintained by the Mozilla project at [PSL].

The term "request", as well as a request's "client", "current url", "method", "target browsing context", and "url list", are defined in [FETCH].

The term "non-HTTP APIs" refers to non-HTTP mechanisms used to set and retrieve cookies, such as a web browser API that exposes cookies to scripts.

The term "top-level navigation" refers to a navigation of a top-level traversable.

3. Overview

This section outlines 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 store state, the origin server includes a Set-Cookie header field in an HTTP response. In subsequent requests, the user agent returns a Cookie request header field to the origin server. The Cookie header field contains cookies the user agent received in previous Set-Cookie header fields. The origin server is free to ignore the Cookie header field or use its contents for an application-defined purpose.

Origin servers MAY send a Set-Cookie response header field with any response. An origin server can include multiple Set-Cookie header fields in a single response. The presence of a Cookie or a Set-Cookie header field does not preclude HTTP caches from storing and reusing a response.

Origin servers and intermediaries MUST NOT combine multiple Set-Cookie header fields into a single header field. The usual mechanism for combining HTTP headers fields (i.e., as defined in Section 5.3 of [HTTP]) might change the semantics of the Set-Cookie header field because the %x2C (",") character is used by Set-Cookie in a way that conflicts with such combining.

For example,

Set-Cookie: a=b;path=/c,d=e

is ambiguous. It could be intended as two cookies, a=b and d=e, or a single cookie with a path of /c,d=e.

User agents MAY ignore Set-Cookie header fields based on response status codes or the user agent's cookie policy (see Section 5.3).

3.1. Examples

Using the Set-Cookie header field, a server can send the user agent a short string in an HTTP response that the user agent will return in future HTTP requests that are within the scope of the cookie. 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 site.example.

== Server -> User Agent ==

Set-Cookie: SID=31d4d96e407aad42; Path=/; Domain=site.example

== User Agent -> Server ==

Cookie: SID=31d4d96e407aad42

As shown in the next example, the server can store multiple cookies at 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 header fields. Notice that the server uses the Secure and HttpOnly attributes to provide additional security protections for the more sensitive session identifier (see Section 4.1.2).

```
   == Server -> User Agent ==

   Set-Cookie: SID=31d4d96e407aad42; Path=/; Secure; HttpOnly
   Set-Cookie: lang=en-US; Path=/; Domain=site.example

   == User Agent -> Server ==

   Cookie: SID=31d4d96e407aad42; lang=en-US
```

Notice that the Cookie header field above contains two cookies, one named SID and one named lang.

Cookie names are case-sensitive, meaning that if a server sends the user agent two Set-Cookie header fields that differ only in their name's case the user agent will store and return both of those cookies in subsequent requests.

```
   == Server -> User Agent ==

   Set-Cookie: SID=31d4d96e407aad42
   Set-Cookie: sid=31d4d96e407aad42

   == User Agent -> Server ==

   Cookie: SID=31d4d96e407aad42; sid=31d4d96e407aad42
```

If the server wishes the user agent to persist the cookie over multiple "sessions" (e.g., user agent restarts), the server can specify an expiration date in the Expires attribute. Note that the user agent 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: SID=31d4d96e407aad42; lang=en-US
```

Finally, to remove a cookie, the server returns a Set-Cookie header field 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 field 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 ==

   Cookie: SID=31d4d96e407aad42
```

3.2. Which Requirements to Implement

The upcoming two sections, Section 4 and Section 5, discuss the set of requirements for two distinct types of implementations. This section is meant to help guide implementers in determining which set of requirements best fits their goals. Choosing the wrong set of requirements could result in a lack of compatibility with other cookie implementations.

It's important to note that being compatible means different things depending on the implementer's goals. These differences have built up over time due to both intentional and unintentional spec changes, spec interpretations, and historical implementation differences.

This section roughly divides implementers of the cookie spec into two types, producers and consumers. These are not official terms and are only used here to help readers develop an intuitive understanding of the use cases.

3.2.1. Cookie Producing Implementations

An implementer should choose Section 4 whenever cookies are created and will be sent to a user agent, such as a web browser. These implementations are frequently referred to as servers by the spec but that term includes anything which primarily produces cookies. Some potential examples:

* Server applications hosting a website or API
* Programming languages or software frameworks that support cookies
* Integrated third-party web applications, such as a business management suite

All these benefit from not only supporting as many user agents as possible but also supporting other servers. This is useful if a cookie is produced by a software framework and is later sent back to a server application which needs to read it. Section 4 advises best practices that help maximize this sense of compatibility.

See Section 3.2.2.1 for more details on programming languages and software frameworks.

3.2.2. Cookie Consuming Implementations

An implementer should choose Section 5 whenever cookies are primarily received from another source. These implementations are referred to as user agents. Some examples:

* Web browsers

* Tools that support stateful HTTP

* Programming languages or software frameworks that support cookies

Because user agents don’t know which servers a user will access, and whether or not that server is following best practices, users agents are advised to implement a more lenient set of requirements and to accept some things that servers are warned against producing. Section 5 advises best practices that help maximize this sense of compatibility.

See Section 3.2.2.1 for more details on programming languages and software frameworks.

3.2.2.1. Programming Languages & Software Frameworks

A programming language or software framework with support for cookies could reasonably be used to create an application that acts as a cookie producer, cookie consumer, or both. Because a developer may want to maximize their compatibility as either a producer or consumer, these languages or frameworks should strongly consider supporting both sets of requirements, Section 4 and Section 5, behind a compatibility mode toggle. This toggle should default to Section 4’s requirements.

Doing so will reduce the chances that a developer’s application can inadvertently create cookies that cannot be read by other servers.

4. Server Requirements

This section describes the syntax and semantics of a well-behaved profile of the Cookie and Set-Cookie header fields.

4.1. Set-Cookie

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

4.1.1. Syntax

Informally, the Set-Cookie response header field contains a cookie, which begins with a name-value-pair, followed by zero or more attribute-value pairs. Servers conforming to this profile MUST NOT send Set-Cookie header fields that deviate from the following grammar:

```
set-cookie        = set-cookie-string
set-cookie-string = BWS cookie-pair *( BWS ";" OWS cookie-av )
cookie-pair       = cookie-name BWS "=" BWS cookie-value
cookie-name       = token
cookie-value      = *cookie-octet / ( DQUOTE *cookie-octet DQUOTE )
cookie-octet      = %x21 / %x23-2B / %x2D-3A / %x3C-5B / %x5D-7E
                      ; US-ASCII characters excluding CTLs,
                      ; whitespace, DQUOTE, comma, semicolon,
                      ; and backslash
token             = <token, defined in [HTTP], Section 5.6.2>

cookie-av         = expires-av / max-age-av / domain-av /
                    path-av / secure-av / httponly-av /
                    samesite-av / extension-av
expires-av        = "Expires" BWS "=" BWS sane-cookie-date
sane-cookie-date  =
    <IMF-fixdate, defined in [HTTP], Section 5.6.7>
max-age-av        = "Max-Age" BWS "=" BWS non-zero-digit *DIGIT
non-zero-digit    = %x31-39
                      ; digits 1 through 9
domain-av         = "Domain" BWS "=" BWS domain-value
domain-value      = <subdomain>
                      ; see details below
path-av           = "Path" BWS "=" BWS path-value
path-value        = *av-octet
secure-av         = "Secure"
httponly-av       = "HttpOnly"
samesite-av       = "SameSite" BWS "=" BWS samesite-value
samesite-value    = "Strict" / "Lax" / "None"
extension-av      = *av-octet
av-octet          = %x20-3A / %x3C-7E
                      ; any CHAR except CTLs or ";"
```

Note that some of the grammatical terms above reference documents that use different grammatical notations than this document (which uses ABNF from [RFC5234]).

Per the grammar above, servers MUST NOT produce nameless cookies (i.e.: an empty cookie-name) as such cookies may be unpredictably serialized by UAs when sent back to the server.

The semantics of the cookie-value are not defined by this document.

To maximize compatibility with user agents, servers that wish to store arbitrary data in a cookie-value SHOULD encode that data, for example, using Base64 [RFC4648].

Per the grammar above, the cookie-value MAY be wrapped in DQUOTE characters. Note that in this case, the initial and trailing DQUOTE characters are not stripped. They are part of the cookie-value, and will be included in Cookie header fields sent to the server.

The domain-value is a subdomain as defined by [RFC1034], Section 3.5, and as enhanced by [RFC1123], Section 2.1. Thus, domain-value is a string of [USASCII] characters, such as an "A-label" as defined in Section 2.3.2.1 of [RFC5890].

The portions of the set-cookie-string produced by the cookie-av term are known as attributes. To maximize compatibility with user agents, servers MUST NOT produce two attributes with the same name in the same set-cookie-string. (See Section 5.7 for how user agents handle this case.)

NOTE: The name of an attribute-value pair is not case-sensitive. So while they are presented here in CamelCase, such as "HttpOnly" or "SameSite", any case is accepted. E.x.: "httponly", "Httponly", "hTTPoNLY", etc.

Servers MUST NOT include more than one Set-Cookie header field in the same response with the same cookie-name. (See Section 5.6 for how user agents handle this case.)

If a server sends multiple responses containing Set-Cookie header fields 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.

NOTE: Some existing user agents differ in their interpretation of two-digit years. To avoid compatibility issues, servers SHOULD use the rfc1123-date format, which requires a four-digit year.

NOTE: Some user agents store and process dates in cookies as 32-bit UNIX time_t values. Implementation bugs in the libraries supporting time_t processing on some systems might cause such user agents to process dates after the year 2038 incorrectly.

4.1.2. Semantics (Non-Normative)

This section describes simplified semantics of the Set-Cookie header field. These semantics are detailed enough to be useful for understanding the most common uses of cookies by servers. The full semantics are described in Section 5.

When the user agent receives a Set-Cookie header field, the user agent stores the cookie together with its attributes. Subsequently, when the user agent makes an HTTP request, the user agent includes the applicable, non-expired cookies in the Cookie header field.

If the user agent receives a new cookie with the same cookie-name, domain-value, and path-value as a cookie that it has already stored, the existing cookie is evicted and replaced with the new cookie. Notice that servers can delete cookies by sending the user agent a new cookie with 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 not, for example, to any subdomains), and it expires at the end of the current session (as defined by the user agent). User agents ignore unrecognized cookie attributes (but not the entire cookie).

4.1.2.1. The Expires Attribute

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

4.1.2.2. The Max-Age Attribute

The Max-Age attribute indicates the maximum lifetime of the cookie, represented as the number of seconds until the cookie expires. The user agent may adjust the specified duration and is not required to retain the cookie for that duration. In fact, user agents often evict cookies due to memory pressure or privacy concerns.

NOTE: Some existing user agents do not support the Max-Age attribute. User agents that do not support the Max-Age attribute ignore the attribute.

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. If a cookie has neither the Max-Age nor the Expires attribute, the user agent will retain the cookie until "the current session is over" (as defined by the user agent).

4.1.2.3. The Domain Attribute

The Domain attribute specifies those hosts to which the cookie will be sent. For example, if the value of the Domain attribute is "site.example", the user agent will include the cookie in the Cookie header field when making HTTP requests to site.example, www.site.example, and www.corp.site.example. (Note that a leading %x2E ("."), if present, is ignored even though that character is not permitted.) If the server omits the Domain attribute, the user agent will return the cookie only to the origin server.

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

The user agent will reject cookies unless the Domain attribute specifies a scope for the cookie that would include the origin server. For example, the user agent will accept a cookie with a Domain attribute of "site.example" or of "foo.site.example" from foo.site.example, but the user agent will not accept a cookie with a Domain attribute of "bar.site.example" or of "baz.foo.site.example".

NOTE: For security reasons, many user agents are configured to reject Domain attributes that correspond to "public suffixes". For example, some user agents will reject Domain attributes of "com" or "co.uk". (See Section 5.7 for more information.)

4.1.2.4. The Path Attribute

The scope of each cookie is limited to a set of paths, controlled by the Path attribute. 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. (See Section 5.1.4 for more details.)

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 %x2F ("/") character is interpreted as a directory separator.

Although seemingly useful for isolating cookies between different paths within a given host, the Path attribute cannot be relied upon for security (see Section 8).

4.1.2.5. The Secure Attribute

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 HTTP over Transport Layer Security (TLS [TLS13]) [HTTP]).

4.1.2.6. The HttpOnly Attribute

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 cookies via non-HTTP APIs.

Note that the HttpOnly attribute is independent of the Secure attribute: a cookie can have both the HttpOnly and the Secure attribute.

4.1.2.7. The SameSite Attribute

The "SameSite" attribute limits the scope of the cookie such that it will only be attached to requests if those requests are same-site, as defined by the algorithm in Section 5.2. For example, requests for https://site.example/sekrit-image will attach same-site cookies if and only if initiated from a context whose "site for cookies" is an origin with a scheme and registered domain of "https" and "site.example" respectively.

If the "SameSite" attribute’s value is "Strict", the cookie will only be sent along with "same-site" requests. If the value is "Lax", the cookie will be sent with same-site requests, and with "cross-site" top-level navigations, as described in Section 5.6.7.1. If the value is "None", the cookie will be sent with same-site and cross-site requests. If the "SameSite" attribute’s value is something other than these three known keywords, the attribute’s value will be subject to a default enforcement mode that is equivalent to "Lax". If a user agent uses "Lax-allowing-unsafe" enforcement (See Section 5.6.7.2) then this default enforcement mode will instead be equivalent to "Lax-allowing-unsafe".

The "SameSite" attribute affects cookie creation as well as delivery. Cookies which assert "SameSite=Lax" or "SameSite=Strict" cannot be set in responses to cross-site subresource requests, or cross-site nested navigations. They can be set along with any top-level navigation, cross-site or otherwise.

4.1.3. Cookie Name Prefixes

Section 8.5 and Section 8.6 of this document spell out some of the drawbacks of cookies' historical implementation. In particular, it is impossible for a server to have confidence that a given cookie was set with a particular set of attributes. In order to provide such confidence in a backwards-compatible way, two common sets of requirements can be inferred from the first few characters of the cookie's name.

The user agent requirements for the prefixes described below are detailed in Section 5.4.

To maximize compatibility with user agents servers SHOULD use prefixes as described below.

4.1.3.1. The "__Secure-" Prefix

If a cookie's name begins with a case-sensitive match for the string __Secure-, then the cookie will have been set with a Secure attribute.

For example, the following Set-Cookie header field would be rejected by a conformant user agent, as it does not have a Secure attribute.

Set-Cookie: __Secure-SID=12345; Domain=site.example

Whereas the following Set-Cookie header field would be accepted if set from a secure origin (e.g. "https://site.example/"), and rejected otherwise:

Set-Cookie: __Secure-SID=12345; Domain=site.example; Secure

4.1.3.2. The "__Host-" Prefix

If a cookie's name begins with a case-sensitive match for the string __Host-, then the cookie will have been set with a Secure attribute, a Path attribute with a value of /, and no Domain attribute.

This combination yields a cookie that hews as closely as a cookie can to treating the origin as a security boundary. The lack of a Domain attribute ensures that the cookie's host-only-flag is true, locking

the cookie to a particular host, rather than allowing it to span subdomains. Setting the Path to / means that the cookie is effective for the entire host, and won't be overridden for specific paths. The Secure attribute ensures that the cookie is unaltered by non-secure origins, and won't span protocols.

Ports are the only piece of the origin model that __Host- cookies continue to ignore.

For example, the following cookies would always be rejected:

```
Set-Cookie: __Host-SID=12345
Set-Cookie: __Host-SID=12345; Secure
Set-Cookie: __Host-SID=12345; Domain=site.example
Set-Cookie: __Host-SID=12345; Domain=site.example; Path=/
Set-Cookie: __Host-SID=12345; Secure; Domain=site.example; Path=/
```

While the following would be accepted if set from a secure origin (e.g. "https://site.example/"), and rejected otherwise:

```
Set-Cookie: __Host-SID=12345; Secure; Path=/
```

4.2. Cookie

4.2.1. Syntax

The user agent sends stored cookies to the origin server in the Cookie header field. If the server conforms to the requirements in Section 4.1 (and the user agent conforms to the requirements in Section 5), the user agent will send a Cookie header field that conforms to the following grammar:

```
cookie        = cookie-string
cookie-string = cookie-pair *( ";" SP cookie-pair )
```

While Section 5.4 of [HTTP] does not define a length limit for header fields it is likely that the web server's implementation does impose a limit; many popular implementations have default limits of 8K. Servers SHOULD avoid setting a large number of large cookies such that the final cookie-string would exceed their header field limit. Not doing so could result in requests to the server failing.

Servers MUST be tolerant of multiple cookie headers. For example, an HTTP/2 [RFC9113] or HTTP/3 [RFC9114] client or intermediary might split a cookie header to improve compression. Servers are free to determine what form this tolerance takes. For example, the server could process each cookie header individually or the server could concatenate all the cookie headers into one and then process that final, single, header. The server should be mindful of any header field limits when deciding which approach to take.

Note: Since intermediaries can modify cookie headers they should also be mindful of common server header field limits in order to avoid sending servers headers that they cannot process. For example, concatenating multiple cookie headers into a single header might exceed a server’s size limit.

4.2.2. Semantics

Each cookie-pair represents a cookie stored by the user agent. The cookie-pair contains the cookie-name and cookie-value the user agent received in the Set-Cookie header field.

Notice that the cookie attributes are not returned. In particular, the server cannot determine from the Cookie field alone when a cookie will expire, for which hosts 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 field are not defined by this document. Servers are expected to imbue these cookies with application-specific semantics.

Although cookies are serialized linearly in the Cookie header field, servers SHOULD NOT rely upon the serialization order. In particular, if the Cookie header field contains two cookies with the same name (e.g., that were set with different Path or Domain attributes), servers SHOULD NOT rely upon the order in which these cookies appear in the header field.

5. User Agent Requirements

This section specifies the Cookie and Set-Cookie header fields in sufficient detail that a user agent implementing these requirements precisely can interoperate with existing servers (even those that do not conform to the well-behaved profile described in Section 4).

A user agent could enforce more restrictions than those specified herein (e.g., restrictions specified by its cookie policy, described in Section 7.2). However, such additional restrictions may reduce the likelihood that a user agent will be able to interoperate with existing servers.

5.1. Subcomponent Algorithms

This section defines some algorithms used by user agents to process specific subcomponents of the Cookie and Set-Cookie header fields.

5.1.1. Dates

The user agent MUST use an algorithm equivalent to the following algorithm to parse a cookie-date. Note that the various boolean flags defined as a part of the algorithm (i.e., found-time, found-day-of-month, found-month, found-year) are initially "not set".

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 )
date-token      = 1*non-delimiter

delimiter       = %x09 / %x20-2F / %x3B-40 / %x5B-60 / %x7B-7E
non-delimiter   = %x00-08 / %x0A-1F / DIGIT / ":" / ALPHA
                  / %x7F-FF
non-digit       = %x00-2F / %x3A-FF

day-of-month    = 1*2DIGIT [ non-digit *OCTET ]
month           = ( "jan" / "feb" / "mar" / "apr" /
                    "may" / "jun" / "jul" / "aug" /
                    "sep" / "oct" / "nov" / "dec" ) *OCTET
year            = 2*4DIGIT [ non-digit *OCTET ]
time            = hms-time [ non-digit *OCTET ]
hms-time        = time-field ":" time-field ":" time-field
time-field      = 1*2DIGIT
```

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

    1. 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.

2. 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.

3. 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.

4. 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.

3. If the year-value is greater than or equal to 70 and less than or equal to 99, increment the year-value by 1900.

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

    1. NOTE: Some existing user agents interpret two-digit years differently.

5. Abort these steps and fail to parse the cookie-date 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,

    * the hour-value is greater than 23,

    * the minute-value is greater than 59, or

    * the second-value is greater than 59.

    (Note that leap seconds cannot be represented in this syntax.)

6. Let the parsed-cookie-date be the date whose day-of-month, month, year, hour, minute, and second (in UTC) are the day-of-month-value, the month-value, the year-value, the hour-value, the minute-value, and the second-value, respectively. If no such date exists, abort these steps and fail to parse the cookie-date.

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

5.1.2. Canonicalized Host Names

A canonicalized host name is the string generated by the following algorithm:

1. Convert the host name to a sequence of individual domain name labels.

2. Convert each label that is not a Non-Reserved LDH (NR-LDH) label, to an A-label (see Section 2.3.2.1 of [RFC5890] for the former and latter).

3. Concatenate the resulting labels, separated by a %x2E (".") character.

5.1.3. Domain Matching

A string domain-matches a given domain string if at least one of the following conditions hold:

* The domain string and the string are identical. (Note that both the domain string and the string will have been canonicalized to lower case at this point.)

* All of the following conditions hold:

  - The domain string is a suffix of the string.

  - The last character of the string that is not included in the domain string is a %x2E (".") character.

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

5.1.4. Paths and Path-Match

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

1. Let uri-path be the path portion of the request-uri if such a portion exists (and empty otherwise).

2. If the uri-path is empty or if the first character of the uri-path is not a %x2F ("/") character, output %x2F ("/") and skip the remaining steps.

3. If the uri-path contains no more than one %x2F ("/") character, output %x2F ("/") and skip the remaining step.

4. Output the characters of the uri-path from the first character up to, but not including, the right-most %x2F ("/").

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

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

  Note that this differs from the rules in [RFC3986] for equivalence of the path component, and hence two equivalent paths can have different cookies.

* The cookie-path is a prefix of the request-path, and the last character of the cookie-path is %x2F ("/").

* 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 %x2F ("/") character.

5.2. "Same-site" and "cross-site" Requests

Two origins are same-site if they satisfy the "same site" criteria defined in [SAMESITE]. A request is "same-site" if the following criteria are true:

1. The request is not the result of a reload navigation triggered through a user interface element (as defined by the user agent; e.g., a request triggered by the user clicking a refresh button on a toolbar).

2. The request’s current url’s origin is same-site with the request’s client’s "site for cookies" (which is an origin), or if the request has no client or the request’s client is null.

Requests which are the result of a reload navigation triggered through a user interface element are same-site if the reloaded document was originally navigated to via a same-site request. A request that is not "same-site" is instead "cross-site".

The request’s client’s "site for cookies" is calculated depending upon its client’s type, as described in the following subsections:

5.2.1. Document-based requests

The URI displayed in a user agent's address bar is the only security context directly exposed to users, and therefore the only signal users can reasonably rely upon to determine whether or not they trust a particular website. The origin of that URI represents the context in which a user most likely believes themselves to be interacting. We'll define this origin, the top-level traversable's active document's origin, as the "top-level origin".

For a document displayed in a top-level traversable, we can stop here: the document's "site for cookies" is the top-level origin.

For container documents, we need to audit the origins of each of a document's ancestor navigables' active documents in order to account for the "multiple-nested scenarios" described in Section 4 of [RFC7034]. A document's "site for cookies" is the top-level origin if and only if the top-level origin is same-site with the document's origin, and with each of the document's ancestor documents' origins. Otherwise its "site for cookies" is an origin set to an opaque origin.

Given a Document (document), the following algorithm returns its "site for cookies":

1. Let top-document be the active document in document's navigable's top-level traversable.
2. Let top-origin be the origin of top-document's URI if top-document's sandboxed origin browsing context flag is set, and top-document's origin otherwise.
3. Let documents be a list consisting of the active documents of document's inclusive ancestor navigables.
4. For each item in documents:
    1. Let origin be the origin of item's URI if item's sandboxed origin browsing context flag is set, and item's origin otherwise.
    2. If origin is not same-site with top-origin, return an origin set to an opaque origin.
5. Return top-origin.

Note: This algorithm only applies when the entire chain of documents from top-document to document are all active.

5.2.2. Worker-based requests

Worker-driven requests aren't as clear-cut as document-driven requests, as there isn't a clear link between a top-level traversable and a worker. This is especially true for Service Workers [SERVICE-WORKERS], which may execute code in the background, without any document visible at all.

Note: The descriptions below assume that workers must be same-origin with the documents that instantiate them. If this invariant changes, we'll need to take the worker's script's URI into account when determining their status.

5.2.2.1. Dedicated and Shared Workers

Dedicated workers are simple, as each dedicated worker is bound to one and only one document. Requests generated from a dedicated worker (via importScripts, XMLHttpRequest, fetch(), etc) define their "site for cookies" as that document's "site for cookies".

Shared workers may be bound to multiple documents at once. As it is quite possible for those documents to have distinct "site for cookies" values, the worker's "site for cookies" will be an origin set to an opaque origin in cases where the values are not all same-site with the worker's origin, and the worker's origin in cases where the values agree.

Given a WorkerGlobalScope (worker), the following algorithm returns its "site for cookies":

1. Let site be worker's origin.
2. For each document in worker's Documents:
    1. Let document-site be document's "site for cookies" (as defined in Section 5.2.1).
    2. If document-site is not same-site with site, return an origin set to an opaque origin.
3. Return site.

5.2.2.2. Service Workers

Service Workers are more complicated, as they act as a completely separate execution context with only tangential relationship to the Document which registered them.

How user agents handle Service Workers may differ, but user agents SHOULD match the [SERVICE-WORKERS] specification.

5.3. Ignoring Set-Cookie Header Fields

User agents MAY ignore Set-Cookie header fields contained in responses with 100-level status codes or based on its cookie policy (see Section 7.2).

All other Set-Cookie header fields SHOULD be processed according to Section 5.6. That is, Set-Cookie header fields contained in responses with non-100-level status codes (including those in responses with 400- and 500-level status codes) SHOULD be processed unless ignored according to the user agent’s cookie policy.

5.4. Cookie Name Prefixes

User agents’ requirements for cookie name prefixes differ slightly from servers’ (Section 4.1.3) in that UAs MUST match the prefix string case-insensitively.

The normative requirements for the prefixes are detailed in the storage model algorithm defined in Section 5.7.

This is because some servers will process cookies case-insensitively, resulting in them unintentionally miscapitalizing and accepting miscapitalized prefixes.

For example, if a server sends the following Set-Cookie header field

Set-Cookie: __SECURE-SID=12345

to a UA which checks prefixes case-sensitively it will accept this cookie and the server would incorrectly believe the cookie is subject the same guarantees as one spelled __Secure-.

Additionally the server is vulnerable to an attacker that purposefully miscapitalizes a cookie in order to impersonate a prefixed cookie. For example, a site already has a cookie __Secure-SID=12345 and by some means an attacker sends the following Set-Cookie header field for the site to a UA which checks prefixes case-sensitively.

Set-Cookie: __SeCuRe-SID=evil

The next time a user visits the site the UA will send both cookies:

Cookie: __Secure-SID=12345; __SeCuRe-SID=evil

The server, being case-insensitive, won't be able to tell the difference between the two cookies allowing the attacker to compromise the site.

To prevent these issues, UAs MUST match cookie name prefixes case-insensitive.

Note: Cookies with different names are still considered separate by UAs. So both __Secure-foo=bar and __secure-foo=baz can exist as distinct cookies simultaneously and both would have the requirements of the __Secure- prefix applied.

The following are examples of Set-Cookie header fields that would be rejected by a conformant user agent.

```
Set-Cookie: __Secure-SID=12345; Domain=site.example
Set-Cookie: __secure-SID=12345; Domain=site.example
Set-Cookie: __SECURE-SID=12345; Domain=site.example
Set-Cookie: __Host-SID=12345
Set-Cookie: __host-SID=12345; Secure
Set-Cookie: __host-SID=12345; Domain=site.example
Set-Cookie: __HOST-SID=12345; Domain=site.example; Path=/
Set-Cookie: __Host-SID=12345; Secure; Domain=site.example; Path=/
Set-Cookie: __host-SID=12345; Secure; Domain=site.example; Path=/
Set-Cookie: __HOST-SID=12345; Secure; Domain=site.example; Path=/
```

Whereas the following Set-Cookie header fields would be accepted if set from a secure origin.

```
Set-Cookie: __Secure-SID=12345; Domain=site.example; Secure
Set-Cookie: __secure-SID=12345; Domain=site.example; Secure
Set-Cookie: __SECURE-SID=12345; Domain=site.example; Secure
Set-Cookie: __Host-SID=12345; Secure; Path=/
Set-Cookie: __host-SID=12345; Secure; Path=/
Set-Cookie: __HOST-SID=12345; Secure; Path=/
```

5.5. Cookie Lifetime Limits

When processing cookies with a specified lifetime, either with the Expires or with the Max-Age attribute, the user agent MUST limit the maximum age of the cookie. The limit SHOULD NOT be greater than 400 days (34560000 seconds) in the future. The RECOMMENDED limit is 400 days in the future, but the user agent MAY adjust the limit (see Section 7.2). Expires or Max-Age attributes that specify a lifetime longer than the limit MUST be reduced to the limit.

5.6. The Set-Cookie Header Field

When a user agent receives a Set-Cookie header field in an HTTP response, the user agent MAY ignore the Set-Cookie header field in its entirety (see Section 5.3).

If the user agent does not ignore the Set-Cookie header field in its entirety, the user agent MUST parse the field-value of the Set-Cookie header field as a set-cookie-string (defined below).

NOTE: The algorithm below is more permissive than the grammar in Section 4.1. For example, the algorithm allows cookie-name to be comprised of cookie-octets instead of being a token as specified in Section 4.1 and the algorithm accommodates some characters that are not cookie-octets according to the grammar in Section 4.1. In addition, the algorithm below also strips leading and trailing whitespace from the cookie name and value (but maintains internal whitespace), whereas the grammar in Section 4.1 forbids whitespace in these positions. User agents use this algorithm so as to interoperate with servers that do not follow the recommendations in Section 4.

NOTE: As set-cookie-string may originate from a non-HTTP API, it is not guaranteed to be free of CTL characters, so this algorithm handles them explicitly. Horizontal tab (%x09) is excluded from the CTL characters that lead to set-cookie-string rejection, as it is considered whitespace, which is handled separately.

NOTE: The set-cookie-string may contain octet sequences that appear percent-encoded as per Section 2.1 of [RFC3986]. However, a user agent MUST NOT decode these sequences and instead parse the individual octets as specified in this algorithm.

A user agent MUST use an algorithm equivalent to the following algorithm to parse a set-cookie-string:

1. If the set-cookie-string contains a %x00-08 / %x0A-1F / %x7F character (CTL characters excluding HTAB): Abort these steps and ignore the set-cookie-string entirely.

2. If the set-cookie-string contains a %x3B (";") character:

    1. The name-value-pair string consists of the characters up to, but not including, the first %x3B (";"), and the unparsed-attributes consist of the remainder of the set-cookie-string (including the %x3B (";") in question).

    Otherwise:

1. 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 %x3D ("=") character, then the name string is empty, and the value string is the value of name-value-pair.

   Otherwise, the (possibly empty) name string consists of the characters up to, but not including, the first %x3D ("=") character, and the (possibly empty) value string consists of the characters after the first %x3D ("=") character.

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

5. If the sum of the lengths of the name string and the value string is more than 4096 octets, abort these steps and ignore the set-cookie-string entirely.

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

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

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

2. Discard the first character of the unparsed-attributes (which will be a %x3B (";") character).

3. If the remaining unparsed-attributes contains a %x3B (";") character:

   1. Consume the characters of the unparsed-attributes up to, but not including, the first %x3B (";") character.

   Otherwise:

   1. 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 %x3D ("=") character:

1. The (possibly empty) attribute-name string consists of the characters up to, but not including, the first %x3D ("=") character, and the (possibly empty) attribute-value string consists of the characters after the first %x3D ("=") character.

Otherwise:

1. The attribute-name string consists of the entire cookie-av string, and the attribute-value string is empty.

5. Remove any leading or trailing WSP characters from the attribute-name string and the attribute-value string.

6. If the attribute-value is longer than 1024 octets, ignore the cookie-av string and return to Step 1 of this algorithm.

7. Process the attribute-name and attribute-value according to the requirements in the following subsections. (Notice that attributes with unrecognized attribute-names are ignored.)

8. Return to Step 1 of this algorithm.

When the user agent finishes parsing the set-cookie-string, the user agent is said to "receive a cookie" from the request-uri with name cookie-name, value cookie-value, and attributes cookie-attribute-list. (See Section 5.7 for additional requirements triggered by receiving a cookie.)

5.6.1. The Expires Attribute

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

1. Let the expiry-time be the result of parsing the attribute-value as cookie-date (see Section 5.1.1).

2. If the attribute-value failed to parse as a cookie date, ignore the cookie-av.

3. Let cookie-age-limit be the maximum age of the cookie (which SHOULD be 400 days in the future or sooner, see Section 5.5).

4. If the expiry-time is more than cookie-age-limit, the user agent MUST set the expiry time to cookie-age-limit in seconds.

5. If the expiry-time is earlier than the earliest date the user agent can represent, the user agent MAY replace the expiry-time with the earliest representable date.

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

5.6.2. The Max-Age Attribute

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

1. If the attribute-value is empty, ignore the cookie-av.

2. If the first character of the attribute-value is neither a DIGIT, nor a "-" character followed by a DIGIT, ignore the cookie-av.

3. If the remainder of attribute-value contains a non-DIGIT character, ignore the cookie-av.

4. Let delta-seconds be the attribute-value converted to a base 10 integer.

5. Let cookie-age-limit be the maximum age of the cookie (which SHOULD be 400 days or less, see Section 5.5).

6. Set delta-seconds to the smaller of its present value and cookie-age-limit.

7. If delta-seconds is less than or equal to zero (0), let expiry-time be the earliest representable date and time. Otherwise, let the expiry-time be the current date and time plus delta-seconds seconds.

8. Append an attribute to the cookie-attribute-list with an attribute-name of Max-Age and an attribute-value of expiry-time.

5.6.3. The Domain Attribute

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

1. Let cookie-domain be the attribute-value.

2. If cookie-domain starts with %x2E ("."), let cookie-domain be cookie-domain without its leading %x2E (".").

3. Convert the cookie-domain to lower case.

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

5.6.4. The Path Attribute

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

1. If the attribute-value is empty or if the first character of the attribute-value is not %x2F ("/"):

    1. Let cookie-path be the default-path.

    Otherwise:

    1. Let cookie-path be the attribute-value.

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

5.6.5. The Secure Attribute

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.

5.6.6. The HttpOnly Attribute

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.

5.6.7. The SameSite Attribute

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

1. Let enforcement be "Default".

2. If cookie-av’s attribute-value is a case-insensitive match for "None", set enforcement to "None".

3. If cookie-av’s attribute-value is a case-insensitive match for "Strict", set enforcement to "Strict".

4. If cookie-av’s attribute-value is a case-insensitive match for "Lax", set enforcement to "Lax".

5. Append an attribute to the cookie-attribute-list with an attribute-name of "SameSite" and an attribute-value of enforcement.

5.6.7.1. "Strict" and "Lax" enforcement

Same-site cookies in "Strict" enforcement mode will not be sent along with top-level navigations which are triggered from a cross-site document context. As discussed in Section 8.8.2, this might or might not be compatible with existing session management systems. In the interests of providing a drop-in mechanism that mitigates the risk of CSRF attacks, developers may set the SameSite attribute in a "Lax" enforcement mode that carves out an exception which sends same-site cookies along with cross-site requests if and only if they are top-level navigations which use a "safe" (in the [HTTP] sense) HTTP method. (Note that a request's method may be changed from POST to GET for some redirects (see Sections 15.4.2 and 15.4.3 of [HTTP]); in these cases, a request's "safe"ness is determined based on the method of the current redirect hop.)

Lax enforcement provides reasonable defense in depth against CSRF attacks that rely on unsafe HTTP methods (like POST), but does not offer a robust defense against CSRF as a general category of attack:

1. Attackers can still pop up new windows or trigger top-level navigations in order to create a "same-site" request (as described in Section 5.2.1), which is only a speedbump along the road to exploitation.

2. Features like <link rel='prerender'> [prerendering] can be exploited to create "same-site" requests without the risk of user detection.

Developers can more completely mitigate CSRF through a session management mechanism such as that described in Section 8.8.2.

5.6.7.2. "Lax-Allowing-Unsafe" enforcement

As discussed in Section 8.8.6, compatibility concerns may necessitate the use of a "Lax-allowing-unsafe" enforcement mode that allows cookies to be sent with a cross-site HTTP request if and only if it is a top-level request, regardless of request method. That is, the "Lax-allowing-unsafe" enforcement mode waives the requirement for the HTTP request's method to be "safe" in the SameSite enforcement step of the retrieval algorithm in Section 5.8.3. (All cookies, regardless of SameSite enforcement mode, may be set for top-level navigations, regardless of HTTP request method, as specified in Section 5.7.)

"Lax-allowing-unsafe" is not a distinct value of the SameSite attribute. Rather, user agents MAY apply "Lax-allowing-unsafe" enforcement only to cookies that did not explicitly specify a SameSite attribute (i.e., those whose same-site-flag was set to "Default" by default). To limit the scope of this compatibility mode, user agents which apply "Lax-allowing-unsafe" enforcement SHOULD restrict the enforcement to cookies which were created recently. Deployment experience has shown a cookie age of 2 minutes or less to be a reasonable limit.

If the user agent uses "Lax-allowing-unsafe" enforcement, it MUST apply the following modification to the retrieval algorithm defined in Section 5.8.3:

Replace the condition in the penultimate bullet point of step 1 of the retrieval algorithm reading

* The HTTP request associated with the retrieval uses a "safe" method.

with

* At least one of the following is true:

  1. The HTTP request associated with the retrieval uses a "safe" method.

  2. The cookie’s same-site-flag is "Default" and the amount of time elapsed since the cookie’s creation-time is at most a duration of the user agent’s choosing.

5.7. Storage Model

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, http-only-flag, and same-site-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. A user agent MAY ignore a received cookie in its entirety. See Section 5.3.

2. If cookie-name is empty and cookie-value is empty, abort these steps and ignore the cookie entirely.

3. If the cookie-name or the cookie-value contains a %x00-08 / %x0A-1F / %x7F character (CTL characters excluding HTAB), abort these steps and ignore the cookie entirely.

4. If the sum of the lengths of cookie-name and cookie-value is more than 4096 octets, abort these steps and ignore the cookie entirely.

5. 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.

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

    1. Set the cookie’s persistent-flag to true.

    2. 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"):

    1. Set the cookie’s persistent-flag to true.

    2. 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:

    1. Set the cookie’s persistent-flag to false.

    2. Set the cookie’s expiry-time to the latest representable date.

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

    1. Let the domain-attribute be the attribute-value of the last attribute in the cookie-attribute-list with both an attribute-name of "Domain" and an attribute-value whose length is no more than 1024 octets. (Note that a leading %x2E ("."), if present, is ignored even though that character is not permitted.)

Otherwise:

1. Let the domain-attribute be the empty string.

8. If the domain-attribute contains a character that is not in the range of [USASCII] characters, abort these steps and ignore the cookie entirely.

9. If the user agent is configured to reject "public suffixes" and the domain-attribute is a public suffix:

    1. If the domain-attribute is identical to the canonicalized request-host:

        1. Let the domain-attribute be the empty string.

        Otherwise:

        1. Abort these steps and ignore the cookie entirely.

    NOTE: This step prevents attacker.example from disrupting the integrity of site.example by setting a cookie with a Domain attribute of "example".

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

    1. If the canonicalized request-host does not domain-match the domain-attribute:

        1. Abort these steps and ignore the cookie entirely.

        Otherwise:

        1. Set the cookie’s host-only-flag to false.

        2. Set the cookie’s domain to the domain-attribute.

    Otherwise:

    1. Set the cookie’s host-only-flag to true.

    2. Set the cookie’s domain to the canonicalized request-host.

11. 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 both an attribute-name of "Path" and an attribute-value whose length is no more than 1024 octets. Otherwise, set the cookie's path to the default-path of the request-uri.

12. 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 the cookie's secure-only-flag to false.

13. If the request-uri does not denote a "secure" connection (as defined by the user agent), and the cookie's secure-only-flag is true, then abort these steps and ignore the cookie entirely.

14. 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 the cookie's http-only-flag to false.

15. If the cookie was received from a "non-HTTP" API and the cookie's http-only-flag is true, abort these steps and ignore the cookie entirely.

16. If the cookie's secure-only-flag is false, and the request-uri does not denote a "secure" connection, then abort these steps and ignore the cookie entirely if the cookie store contains one or more cookies that meet all of the following criteria:

    1. Their name matches the name of the newly-created cookie.

    2. Their secure-only-flag is true.

    3. Their domain domain-matches the domain of the newly-created cookie, or vice-versa.

    4. The path of the newly-created cookie path-matches the path of the existing cookie.

    Note: The path comparison is not symmetric, ensuring only that a newly-created, non-secure cookie does not overlay an existing secure cookie, providing some mitigation against cookie-fixing attacks. That is, given an existing secure cookie named 'a' with a path of '/login', a non-secure cookie named 'a' could be set for a path of '/' or '/foo', but not for a path of '/login' or '/login/en'.

17. If the cookie-attribute-list contains an attribute with an attribute-name of "SameSite", and an attribute-value of "Strict", "Lax", or "None", set the cookie's same-site-flag to the attribute-value of the last attribute in the cookie-attribute-list with an attribute-name of "SameSite". Otherwise, set the cookie's same-site-flag to "Default".

18. If the cookie's same-site-flag is not "None":

    1. If the cookie was received from a "non-HTTP" API, and the API was called from a navigable's active document whose "site for cookies" is not same-site with the top-level origin, then abort these steps and ignore the newly created cookie entirely.

    2. If the cookie was received from a "same-site" request (as defined in Section 5.2), skip the remaining substeps and continue processing the cookie.

    3. If the cookie was received from a request which is navigating a top-level traversable [HTML] (e.g. if the request's "reserved client" is either null or an environment whose "target browsing context"'s navigable is a top-level traversable), skip the remaining substeps and continue processing the cookie.

       Note: Top-level navigations can create a cookie with any SameSite value, even if the new cookie wouldn't have been sent along with the request had it already existed prior to the navigation.

    4. Abort these steps and ignore the newly created cookie entirely.

19. If the cookie's "same-site-flag" is "None", abort these steps and ignore the cookie entirely unless the cookie's secure-only-flag is true.

20. If the cookie-name begins with a case-insensitive match for the string "__Secure-", abort these steps and ignore the cookie entirely unless the cookie's secure-only-flag is true.

21. If the cookie-name begins with a case-insensitive match for the string "__Host-", abort these steps and ignore the cookie entirely unless the cookie meets all the following criteria:

    1. The cookie's secure-only-flag is true.

2. The cookie’s host-only-flag is true.

3. The cookie-attribute-list contains an attribute with an attribute-name of "Path", and the cookie’s path is /.

22. If the cookie-name is empty and either of the following conditions are true, abort these steps and ignore the cookie entirely:

    * the cookie-value begins with a case-insensitive match for the string "__Secure-"

    * the cookie-value begins with a case-insensitive match for the string "__Host-"

23. If the cookie store contains a cookie with the same name, domain, host-only-flag, and path as the newly-created cookie:

    1. Let old-cookie be the existing cookie with the same name, domain, host-only-flag, 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 a "non-HTTP" API and the old-cookie’s http-only-flag is true, 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.

24. Insert the newly-created cookie into the cookie store.

A cookie is "expired" if the cookie has an expiry date in the past.

The user agent MUST evict all expired cookies from the cookie store if, at any time, an expired cookie exists in the cookie store.

At any time, the user agent MAY "remove excess cookies" from the cookie store if the number of cookies sharing a domain field exceeds some implementation-defined upper bound (such as 50 cookies).

At any time, the user agent MAY "remove excess cookies" from the cookie store if the cookie store exceeds some predetermined upper bound (such as 3000 cookies).

When the user agent removes excess cookies from the cookie store, the user agent MUST evict cookies in the following priority order:

1. Expired cookies.

2. Cookies whose secure-only-flag is false, and which share a domain field with more than a predetermined number of other cookies.

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

4. All cookies.

If two cookies have the same removal priority, the user agent MUST evict the cookie with the earliest last-access-time 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.

5.8. Retrieval Model

This section defines how cookies are retrieved from a cookie store in the form of a cookie-string. A "retrieval" is any event which requires generating a cookie-string. For example, a retrieval may occur in order to build a Cookie header field for an HTTP request, or may be required in order to return a cookie-string from a call to a "non-HTTP" API that provides access to cookies. A retrieval has an associated URI, same-site status, and type, which are defined below depending on the type of retrieval.

5.8.1. The Cookie Header Field

The user agent includes stored cookies in the Cookie HTTP request header field.

A user agent MAY omit the Cookie header field in its entirety. For example, the user agent might wish to block sending cookies during "third-party" requests from setting cookies (see Section 7.1).

If the user agent does attach a Cookie header field to an HTTP request, the user agent MUST generate a single cookie-string and the user agent MUST compute the cookie-string following the algorithm defined in Section 5.8.3, where the retrieval's URI is the request-uri, the retrieval's same-site status is computed for the HTTP request as defined in Section 5.2, and the retrieval's type is "HTTP".

Note: Previous versions of this specification required that only one Cookie header field be sent in requests. This is no longer a requirement. While this specification requires that a single cookie-string be produced, some user agents may split that string across multiple cookie header fields. For examples, see Section 8.2.3 of [RFC9113] and Section 4.2.1 of [RFC9114].

5.8.2. Non-HTTP APIs

The user agent MAY implement "non-HTTP" APIs that can be used to access stored cookies.

A user agent MAY return an empty cookie-string in certain contexts, such as when a retrieval occurs within a third-party context (see Section 7.1).

If a user agent does return cookies for a given call to a "non-HTTP" API with an associated Document, then the user agent MUST compute the cookie-string following the algorithm defined in Section 5.8.3, where the retrieval's URI is defined by the caller (see [DOM-DOCUMENT-COOKIE]), the retrieval's same-site status is "same-site" if the Document's "site for cookies" is same-site with the top-level origin as defined in Section 5.2.1 (otherwise it is "cross-site"), and the retrieval's type is "non-HTTP".

5.8.3. Retrieval Algorithm

Given a cookie store and a retrieval, the following algorithm returns a cookie-string from a given cookie store.

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

    * Either:

        - The cookie's host-only-flag is true and the canonicalized host of the retrieval's URI is identical to the cookie's domain.

        Or:

        - The cookie's host-only-flag is false and the canonicalized host of the retrieval's URI domain-matches the cookie's domain.

        NOTE: (For user agents configured to reject "public suffixes") It's possible that the public suffix list was changed since a cookie was created. If this change results in a cookie's

domain becoming a public suffix then that cookie is considered invalid as it would have been rejected during creation (See Section 5.7 step 9). User agents should be careful to avoid retrieving these invalid cookies even if they domain-match the host of the retrieval's URI.

* The retrieval's URI's path path-matches the cookie's path.

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

  NOTE: The notion of a "secure" connection is not defined by this document. Typically, user agents consider a connection secure if the connection makes use of transport-layer security, such as SSL or TLS [TLS13], or if the host is trusted. For example, most user agents consider "https" to be a scheme that denotes a secure protocol and "localhost" to be trusted host.

* If the cookie's http-only-flag is true, then exclude the cookie if the retrieval's type is "non-HTTP".

* If the cookie's same-site-flag is not "None" and the retrieval's same-site status is "cross-site", then exclude the cookie unless all of the following conditions are met:

  - The retrieval's type is "HTTP".

  - The same-site-flag is "Lax" or "Default".

  - The HTTP request associated with the retrieval uses a "safe" method.

  - The target browsing context of the HTTP request associated with the retrieval is the active browsing context or a top-level traversable.

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, and, historically, there have been servers that (erroneously) depended on this order.

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 cookies' name is not empty, output the cookie's name followed by the %x3D ("=") character.

    2. If the cookies' value is not empty, output the cookie's value.

    3. If there is an unprocessed cookie in the cookie-list, output the characters %x3B and %x20 ("; ").

6. Implementation Considerations

6.1. Limits

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 50 cookies per domain.

* At least 3000 cookies total.

User agents MAY limit the maximum number of cookies they store, and may evict any cookie at any time (whether at the request of the user or due to implementation limitations).

Note that a limit on the maximum number of cookies also limits the total size of the stored cookies, due to the length limits which MUST be enforced in Section 5.6.

Servers SHOULD use as few and as small cookies as possible to avoid reaching these implementation limits, minimize network bandwidth due to the Cookie header field being included in every request, and to avoid reaching server header field limits (See Section 4.2.1).

Servers SHOULD gracefully degrade if the user agent fails to return one or more cookies in the Cookie header field because the user agent might evict any cookie at any time.

6.2. Application Programming Interfaces

One reason the Cookie and Set-Cookie header fields use such esoteric syntax is that many platforms (both in servers and user agents) provide a string-based application programming interface (API) to cookies, requiring application-layer programmers to generate and parse the syntax used by the Cookie and Set-Cookie header fields, which many programmers have done incorrectly, resulting in interoperability problems.

Instead of providing string-based APIs to cookies, platforms 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 an abstract "Date" object instead of a serialized date string.

7. Privacy Considerations

Cookies' primary privacy risk is their ability to correlate user activity. This can happen on a single site, but is most problematic when activity is tracked across different, seemingly unconnected Web sites to build a user profile.

Over time, this capability (warned against explicitly in [RFC2109] and all of its successors) has become widely used for varied reasons including:

* authenticating users across sites,
* assembling information on users,
* protecting against fraud and other forms of undesirable traffic,
* targeting advertisements at specific users or at users with specified attributes,
* measuring how often ads are shown to users, and
* recognizing when an ad resulted in a change in user behavior.

While not every use of cookies is necessarily problematic for privacy, their potential for abuse has become a widespread concern in the Internet community and broader society. In response to these concerns, user agents have actively constrained cookie functionality in various ways (as allowed and encouraged by previous specifications), while avoiding disruption to features they judge desirable for the health of the Web.

It is too early to declare consensus on which specific mechanism(s) should be used to mitigate cookies' privacy impact; user agents' ongoing changes to how they are handled are best characterised as experiments that can provide input into that eventual consensus.

Instead, this document describes limited, general mitigations against the privacy risks associated with cookies that enjoy wide deployment at the time of writing. It is expected that implementations will continue to experiment and impose stricter, more well-defined limitations on cookies over time. Future versions of this document might codify those mechanisms based upon deployment experience. If functions that currently rely on cookies can be supported by separate, targeted mechanisms, they might be documented in separate specifications and stricter limitations on cookies might become feasible.

Note that cookies are not the only mechanism that can be used to track users across sites, so while these mitigations are necessary to improve Web privacy, they are not sufficient on their own.

7.1. Third-Party Cookies

A "third-party" or cross-site cookie is one that is associated with embedded content (such as scripts, images, stylesheets, frames) that is obtained from a different server than the one that hosts the primary resource (usually, the Web page that the user is viewing). Third-party cookies are often used to correlate users' activity on different sites.

Because of their inherent privacy issues, most user agents now limit third-party cookies in a variety of ways. Some completely block third-party cookies by refusing to process third-party Set-Cookie header fields and refusing to send third-party Cookie header fields. Some partition cookies based upon the first-party context, so that different cookies are sent depending on the site being browsed. Some block cookies based upon user agent cookie policy and/or user controls.

While this document does not endorse or require a specific approach, it is RECOMMENDED that user agents adopt a policy for third-party cookies that is as restrictive as compatibility constraints permit. Consequently, resources cannot rely upon third-party cookies being treated consistently by user agents for the foreseeable future.

7.2. Cookie Policy

User agents MAY enforce a cookie policy consisting of restrictions on how cookies may be used or ignored (see Section 5.3).

A cookie policy may govern which domains or parties, as in first and third parties (see Section 7.1), for which the user agent will allow cookie access. The policy can also define limits on cookie size, cookie expiry (see Section 5.5), and the number of cookies per domain or in total.

The recommended cookie expiry upper limit is 400 days. User agents may set a lower limit to enforce shorter data retention timelines, or set the limit higher to support longer retention when appropriate (e.g., server-to-server communication over HTTPS).

The goal of a restrictive cookie policy is often to improve security or privacy. User agents often allow users to change the cookie policy (see Section 7.3).

7.3. User Controls

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 agents include a user interface element that lets users examine the cookies stored in their 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 field in outbound HTTP requests and the user agent MUST NOT process Set-Cookie header fields in inbound HTTP responses.

User agents MAY offer a way to change the cookie policy (see Section 7.2).

User agents MAY 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 popular user agents expose this functionality via "private browsing" mode [Aggarwal2010].

7.4. Expiration Dates

Although servers can set the expiration date for cookies to the distant future, most user agents do not actually retain cookies for multiple decades. Rather than choosing gratuitously long expiration periods, servers SHOULD promote user privacy by selecting reasonable cookie expiration periods based on the purpose of the cookie. For example, a typical session identifier might reasonably be set to expire in two weeks.

8. Security Considerations

8.1. Overview

Cookies have a number of security pitfalls. This section overviews a few of the more salient issues.

In particular, cookies encourage developers to rely on ambient authority for authentication, often becoming vulnerable to attacks such as cross-site request forgery [CSRF]. Also, 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, cookies do not provide confidentiality or integrity from network attackers, even when used in conjunction with HTTPS.

8.2. Ambient Authority

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 or HTML forms). When issuing those requests, user agents attach cookies even if the remote party does not know the contents of the cookies, potentially letting the remote party 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 application of these principles can lead to more robust security.

8.3. Clear Text

Unless sent over a secure channel (such as TLS [TLS13]), the information in the Cookie and Set-Cookie header fields is transmitted in the clear.

1. All sensitive information conveyed in these header fields is exposed to an eavesdropper.

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

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

Servers SHOULD encrypt and sign the contents of cookies (using whatever format the server desires) 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 and Set-Cookie header fields only over a secure channel. When using cookies over a secure channel, servers SHOULD set the Secure attribute (see Section 4.1.2.5) for every cookie. If a server does not set the Secure attribute, the protection provided by the secure channel will be largely moot.

For example, consider a webmail server that stores a session identifier in a cookie and is typically accessed over HTTPS. If the server does not set the Secure attribute on its cookies, an active network attacker can intercept any outbound HTTP request from the user agent and redirect that request to the webmail server over HTTP. Even if the webmail server is not listening for HTTP connections, the user agent will still include cookies in the request. The active network attacker can intercept these cookies, replay them against the server, and learn the contents of the user's email. If, instead, the server had set the Secure attribute on its cookies, the user agent would not have included the cookies in the clear-text request.

8.4. Session Identifiers

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

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 be available to other schemes, such as ftp and gopher. Although this lack of isolation by scheme is most apparent in 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 requirements for processing cookies themselves (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 cookies 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

Cookies do not provide integrity guarantees for sibling domains (and their subdomains). For example, consider foo.site.example and bar.site.example. The foo.site.example server can set a cookie with a Domain attribute of "site.example" (possibly overwriting an existing "site.example" cookie set by bar.site.example), and the user agent will include that cookie in HTTP requests to bar.site.example. In the worst case, bar.site.example will be unable to distinguish this cookie from a cookie it set itself. The foo.site.example server might be able to leverage this ability to mount an attack against bar.site.example.

Even though the Set-Cookie header field 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 field. For example, an HTTP response to a request for http://site.example/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 to store security-sensitive information.

An active network attacker can also inject cookies into the Cookie header field sent to https://site.example/ by impersonating a response from http://site.example/ and injecting a Set-Cookie header field. The HTTPS server at site.example 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 site.example even if site.example uses HTTPS exclusively.

Servers can partially mitigate these attacks by encrypting and signing the contents of their cookies, or by naming the cookie with the __Secure- prefix. However, using cryptography does not mitigate the issue completely because an attacker can replay a cookie he or she received from the authentic site.example 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 a 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

Cookies rely 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.

8.8. SameSite Cookies

8.8.1. Defense in depth

"SameSite" cookies offer a robust defense against CSRF attack when deployed in strict mode, and when supported by the client. It is, however, prudent to ensure that this designation is not the extent of a site's defense against CSRF, as same-site navigations and submissions can certainly be executed in conjunction with other attack vectors such as cross-site scripting or abuse of page redirections.

Understanding how and when a request is considered same-site is also important in order to properly design a site for SameSite cookies. For example, if a cross-site top-level request is made to a sensitive page that request will be considered cross-site and SameSite=Strict cookies won't be sent; that page's sub-resources requests, however, are same-site and would receive SameSite=Strict cookies. Sites can avoid inadvertently allowing access to these sub-resources by returning an error for the initial page request if it doesn't include the appropriate cookies.

Developers are strongly encouraged to deploy the usual server-side defenses (CSRF tokens, ensuring that "safe" HTTP methods are idempotent, etc) to mitigate the risk more fully.

Additionally, client-side techniques such as those described in [app-isolation] may also prove effective against CSRF, and are certainly worth exploring in combination with "SameSite" cookies.

8.8.2. Top-level Navigations

Setting the SameSite attribute in "strict" mode provides robust defense in depth against CSRF attacks, but has the potential to confuse users unless sites' developers carefully ensure that their cookie-based session management systems deal reasonably well with top-level navigations.

Consider the scenario in which a user reads their email at MegaCorp Inc's webmail provider https://site.example/. They might expect that clicking on an emailed link to https://projects.example/secret/ project would show them the secret project that they're authorized to see, but if https://projects.example has marked their session cookies as SameSite=Strict, then this cross-site navigation won't send them along with the request. https://projects.example will render a 404 error to avoid leaking secret information, and the user will be quite confused.

Developers can avoid this confusion by adopting a session management system that relies on not one, but two cookies: one conceptually granting "read" access, another granting "write" access. The latter could be marked as SameSite=Strict, and its absence would prompt a reauthentication step before executing any non-idempotent action. The former could be marked as SameSite=Lax, in order to allow users access to data via top-level navigation, or SameSite=None, to permit access in all contexts (including cross-site embedded contexts).

8.8.3. Mashups and Widgets

The Lax and Strict values for the SameSite attribute are inappropriate for some important use-cases. In particular, note that content intended for embedding in cross-site contexts (social networking widgets or commenting services, for instance) will not have access to same-site cookies. Cookies which are required in these situations should be marked with SameSite=None to allow access in cross-site contexts.

Likewise, some forms of Single-Sign-On might require cookie-based authentication in a cross-site context; these mechanisms will not function as intended with same-site cookies and will also require SameSite=None.

8.8.4. Server-controlled

SameSite cookies in and of themselves don’t do anything to address the general privacy concerns outlined in Section 7.1 of [RFC6265]. The "SameSite" attribute is set by the server, and serves to mitigate the risk of certain kinds of attacks that the server is worried about. The user is not involved in this decision. Moreover, a number of side-channels exist which could allow a server to link distinct requests even in the absence of cookies (for example, connection and/or socket pooling between same-site and cross-site requests).

8.8.5. Reload navigations

Requests issued for reloads triggered through user interface elements (such as a refresh button on a toolbar) are same-site only if the reloaded document was originally navigated to via a same-site request. This differs from the handling of other reload navigations, which are always same-site if top-level, since the source navigable’s active document is precisely the document being reloaded.

This special handling of reloads triggered through a user interface element avoids sending SameSite cookies on user-initiated reloads if they were withheld on the original navigation (i.e., if the initial navigation were cross-site). If the reload navigation were instead considered same-site, and sent all the initially withheld SameSite cookies, the security benefits of withholding the cookies in the first place would be nullified. This is especially important given that the absence of SameSite cookies withheld on a cross-site navigation request may lead to visible site breakage, prompting the user to trigger a reload.

For example, suppose the user clicks on a link from https://attacker.example/ to https://victim.example/. This is a cross-site request, so SameSite=Strict cookies are withheld. Suppose this causes https://victim.example/ to appear broken, because the site only displays its sensitive content if a particular SameSite cookie is present in the request. The user, frustrated by the unexpectedly broken site, presses refresh on their browser’s toolbar. To now consider the reload request same-site and send the initially withheld SameSite cookie would defeat the purpose of withholding it in the first place, as the reload navigation triggered through the user interface may replay the original (potentially malicious) request. Thus, the reload request should be considered cross-site, like the request that initially navigated to the page.

Because requests issued for, non-user initiated, reloads attach all SameSite cookies, developers should be careful and thoughtful about when to initiate a reload in order to avoid a CSRF attack. For example, the page could only initiate a reload if a CSRF token is present on the initial request.

8.8.6. Top-level requests with "unsafe" methods

The "Lax" enforcement mode described in Section 5.6.7.1 allows a cookie to be sent with a cross-site HTTP request if and only if it is a top-level navigation with a "safe" HTTP method. Implementation experience shows that this is difficult to apply as the default behavior, as some sites may rely on cookies not explicitly specifying a SameSite attribute being included on top-level cross-site requests with "unsafe" HTTP methods (as was the case prior to the introduction of the SameSite attribute).

For example, the concluding step of a login flow may involve a cross-site top-level POST request to an endpoint; this endpoint expects a recently created cookie containing transactional state information, necessary to securely complete the login. For such a cookie, "Lax" enforcement is not appropriate, as it would cause the cookie to be excluded due to the unsafe HTTP request method, resulting in an unrecoverable failure of the whole login flow.

The "Lax-allowing-unsafe" enforcement mode described in Section 5.6.7.2 retains some of the protections of "Lax" enforcement (as compared to "None") while still allowing recently created cookies to be sent cross-site with unsafe top-level requests.

As a more permissive variant of "Lax" mode, "Lax-allowing-unsafe" mode necessarily provides fewer protections against CSRF. Ultimately, the provision of such an enforcement mode should be seen as a temporary, transitional measure to ease adoption of "Lax" enforcement by default.

9. IANA Considerations

9.1. Cookie

The HTTP Field Name Registry (see [HttpFieldNameRegistry]) needs to be updated with the following registration:

Header field name: Cookie

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document: this specification (Section 5.8.1)

9.2. Set-Cookie

The permanent message header field registry (see [HttpFieldNameRegistry]) needs to be updated with the following registration:

Header field name: Set-Cookie

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document: this specification (Section 5.6)

9.3. "Cookie Attributes" Registry

IANA is requested to create the "Cookie Attributes" registry, defining the name space of attributes used to control cookies' behavior. The registry should be maintained in a new registry group called "Hypertext Transfer Protocol (HTTP) Cookie Attributes" at https://www.iana.org/assignments/cookie-attribute-names (https://www.iana.org/assignments/cookie-attribute-names).

9.3.1. Procedure

Each registered attribute name is associated with a description, and a reference detailing how the attribute is to be processed and stored.

New registrations happen on a "RFC Required" basis (see Section 4.7 of [RFC8126]). The attribute to be registered MUST match the extension-av syntax defined in Section 4.1.1. Note that attribute names are generally defined in CamelCase, but technically accepted case-insensitively.

9.3.2. Registration

The "Cookie Attributes" registry should be created with the registrations below:

Name	Reference
Domain	Section 4.1.2.3 of this document
Expires	Section 4.1.2.1 of this document
HttpOnly	Section 4.1.2.6 of this document
Max-Age	Section 4.1.2.2 of this document
Path	Section 4.1.2.4 of this document
SameSite	Section 4.1.2.7 of this document
Secure	Section 4.1.2.5 of this document

Table 1

10. References

10.1. Normative References

[DOM-DOCUMENT-COOKIE]
WHATWG, "HTML - Living Standard", 18 May 2021, <https://html.spec.whatwg.org/#dom-document-cookie>.

[FETCH] van Kesteren, A., "Fetch", n.d., <https://fetch.spec.whatwg.org/>.

[HTML] Hickson, I., Pieters, S., van Kesteren, A., J辰genstedt, P., and D. Denicola, "HTML", n.d., <https://html.spec.whatwg.org/>.

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/rfc/rfc1034>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <https://www.rfc-editor.org/rfc/rfc1123>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC4790] Newman, C., Duerst, M., and A. Gulbrandsen, "Internet Application Protocol Collation Registry", RFC 4790, DOI 10.17487/RFC4790, March 2007, <https://www.rfc-editor.org/rfc/rfc4790>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/rfc/rfc5234>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/rfc/rfc5890>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <https://www.rfc-editor.org/rfc/rfc6454>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[SAMESITE] WHATWG, "HTML - Living Standard", 26 January 2021, <https://html.spec.whatwg.org/#same-site>.

[USASCII] American National Standards Institute, "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

10.2. Informative References

[Aggarwal2010]
Aggarwal, G., Burzstein, E., Jackson, C., and D. Boneh, "An Analysis of Private Browsing Modes in Modern Browsers", 2010, <http://www.usenix.org/events/sec10/tech/full_papers/Aggarwal.pdf>.

[app-isolation]
Chen, E., Bau, J., Reis, C., Barth, A., and C. Jackson, "App Isolation - Get the Security of Multiple Browsers with Just One", 2011, <http://www.collinjackson.com/research/papers/appisolation.pdf>.

[CSRF] Barth, A., Jackson, C., and J. Mitchell, "Robust Defenses for Cross-Site Request Forgery", DOI 10.1145/1455770.1455782, ISBN 978-1-59593-810-7, ACM CCS '08: Proceedings of the 15th ACM conference on Computer and communications security (pages 75-88), October 2008, <http://portal.acm.org/citation.cfm?id=1455770.1455782>.

[HttpFieldNameRegistry]
"Hypertext Transfer Protocol (HTTP) Field Name Registry", n.d., <https://www.iana.org/assignments/http-fields/>.

[prerendering]
Bentzel, C., "Chrome Prerendering", n.d., <https://www.chromium.org/developers/design-documents/prerender>.

[PSL] "Public Suffix List", n.d., <https://publicsuffix.org/list/>.

[RFC2109] Kristol, D. and L. Montulli, "HTTP State Management Mechanism", RFC 2109, DOI 10.17487/RFC2109, February 1997, <https://www.rfc-editor.org/rfc/rfc2109>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/rfc/rfc6265>.

[RFC7034] Ross, D. and T. Gondrom, "HTTP Header Field X-Frame-Options", RFC 7034, DOI 10.17487/RFC7034, October 2013, <https://www.rfc-editor.org/rfc/rfc7034>.

[RFC9113] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/rfc/rfc9113>.

[RFC9114] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/rfc/rfc9114>.

[SERVICE-WORKERS]
Archibald, J. and M. Kruisselbrink, "Service Workers", n.d., <https://www.w3.org/TR/service-workers/>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

Appendix A. Changes from RFC 6265

* Adds the same-site concept and the SameSite attribute. (Section 5.2 and Section 4.1.2.7)

* Introduces cookie prefixes and prohibits nameless cookies from setting a value that would mimic a cookie prefix. (Section 4.1.3 and Section 5.7)

* Prohibits non-secure origins from setting cookies with a Secure flag or overwriting cookies with this flag. (Section 5.7)

* Limits maximum cookie size. (Section 5.7)

* Limits maximum values for max-age and expire. (Section 5.6.1 and Section 5.6.2)

* Includes the host-only-flag as part of a cookie’s uniqueness computation. (Section 5.7)

* Considers potentially trustworthy origins as "secure". (Section 5.7)

* Improves cookie syntax

  - Treats Set-Cookie: token as creating the cookie ("", "token"). (Section 5.6)

  - Rejects cookies without a name nor value. (Section 5.7)

  - Specifies how to serialize a nameless/valueless cookie. (Section 5.8.3)

  - Adjusts ABNF for cookie-pair and the Cookie header production to allow for spaces. (Section 4.1.1)

  - Explicitly handle control characters. (Section 5.6 and Section 5.7)

  - Specifies how to handle empty domain attributes. (Section 5.7)

  - Requires ASCII characters for the domain attribute. (Section 5.7)

* Refactors cookie retrieval algorithm to support non-HTTP APIs. (Section 5.8.2)

* Specifies that the Set-Cookie line should not be decoded. (Section 5.6)

* Adds an advisory section to assist implementers in deciding which requirements to implement. (Section 3.2)

* Advises against sending invalid cookies due to public suffix list changes. (Section 5.8.3)

* Removes the single cookie header requirement. (Section 5.8.1)

* Address errata 3444 by updating the path-value andextension-av grammar, errata 4148 by updating the day-of-month, year, and time grammar, and errata 3663 by adding the requested note. (Section 4.1 and Section 5.1.4)

Acknowledgements

RFC 6265 was written by Adam Barth. This document is an update of RFC 6265, adding features and aligning the specification with the reality of today’s deployments. Here, we’re standing upon the shoulders of a giant since the majority of the text is still Adam’s.

Thank you to both Lily Chen and Steven Englehardt, editors emeritus, for their significant contributions improving this draft.

Authors’ Addresses

Steven Bingler (editor)
Google LLC
Email: bingler@google.com

Mike West (editor)
Google LLC
Email: mkwst@google.com
URI: https://mikewest.org/

John Wilander (editor)
Apple, Inc
Email: wilander@apple.com