HTTP Working Group
Internet-Draft
Updates: 6265 (if approved)
Intended status: Standards Track
Expires: December 22, 2016

M. West
Google, Inc
M. Goodwin
Mozilla

June 20, 2016

Same-Site Cookies
draft-ietf-httpbis-cookie-same-site-00

Abstract

This document updates RFC6265 by defining a "SameSite" attribute which allows servers to assert that a cookie ought not to be sent along with cross-site requests. This assertion allows user agents to mitigate the risk of cross-origin information leakage, and provides some protection against cross-site request forgery attacks.

Note to Readers

Discussion of this draft takes place on the HTTP working group mailing list (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 http://httpwg.github.io/ ; source code and issues list for this draft can be found at https://github.com/httpwg/http-extensions/labels/cookie-same-site .

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 http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 22, 2016.

Table of Contents

1. Introduction

Section 8.2 of [RFC6265] eloquently notes that cookies are a form of ambient authority, attached by default to requests the user agent sends on a user's behalf. Even when an attacker doesn't know the contents of a user's cookies, she can still execute commands on the user's behalf (and with the user's authority) by asking the user agent to send HTTP requests to unwary servers.

Here, we update [RFC6265] with a simple mitigation strategy that allows servers to declare certain cookies as "same-site", meaning they should not be attached to "cross-site" requests (as defined in section 2.1).

Note that the mechanism outlined here is backwards compatible with the existing cookie syntax. Servers may serve these cookies to all user agents; those that do not support the "SameSite" attribute will simply store a cookie which is attached to all relevant requests, just as they do today.

1.1. Goals

These cookies are intended to provide a solid layer of defense-in-depth against attacks which require embedding an authenticated request into an attacker-controlled context:

1. Timing attacks which yield cross-origin information leakage (such as those detailed in [pixel-perfect]) can be substantially mitigated by setting the "SameSite" attribute on authentication cookies. The attacker will only be able to embed unauthenticated resources, as embedding mechanisms such as "<iframe>" will yield cross-site requests.
2. Cross-site script inclusion (XSSI) attacks are likewise mitigated by setting the "SameSite" attribute on authentication cookies. The attacker will not be able to include authenticated resources via "<script>" or "<link>", as these embedding mechanisms will likewise yield cross-site requests.
3. Cross-site request forgery (CSRF) attacks which rely on top-level navigation (HTML "<form>" POSTs, for instance) can also be mitigated by treating these navigational requests as "cross-site".
4. Same-site cookies have some marginal value for policy or regulatory purposes, as cookies which are not delivered with cross-site requests cannot be directly used for tracking purposes. It may be valuable for an origin to assert that its cookies should not be sent along with cross-site requests in order to limit its exposure to non-technical risk.

1.2. Examples

Same-site cookies are set via the "SameSite" attribute in the "Set-Cookie" header field. That is, given a server's response to a user agent which contains the following header field:

```
Set-Cookie: SID=31d4d96e407aad42; SameSite=Strict
```

Subsequent requests from that user agent can be expected to contain the following header field if and only if both the requested resource and the resource in the top-level browsing context match the cookie.

2. Terminology and notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

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

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 terms "active document", "ancestor browsing context", "browsing context", "document", "WorkerGlobalScope", "sandboxed origin browsing context flag", "parent browsing context", "the worker's Documents", "nested browsing context", and "top-level browsing context" 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 4.2.1 of [RFC7231].

The term "public suffix" is defined in a note in Section 5.3 of [RFC6265] as "a domain that is controlled by a public registry". For example, "example.com"'s public suffix is "com". User agents SHOULD use an up-to-date public suffix list, such as the one maintained by Mozilla at [PSL].

An origin's "registrable domain" is the origin's host's public suffix plus the label to its left. That is, "https://www.example.com"'s registrable domain is "example.com". This concept is defined more rigorously in [PSL].

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

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

A request is "same-site" if its target's URI's origin's registrable domain is an exact match for the request's initiator's "site for cookies", and "cross-site" otherwise. To be more precise, for a given request ("request"), the following algorithm returns "same-site" or "cross-site":

1. If "request"'s client is "null", return "same-site".
2. Let "site" be "request"'s client's "site for cookies" (as defined in the following sections).
3. Let "target" be the registrable domain of "request"'s current url.
4. If "site" is an exact match for "target", return "same-site".
5. Return "cross-site".

2.1.1. 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 registrable domain of that URI's origin represents the context in which a user most likely believes themselves to be interacting. We'll label this domain the "top-level site".

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

For documents which are displayed in nested browsing contexts, we need to audit the origins of each of a document's ancestor browsing contexts' active documents in order to account for the "multiple-nested scenarios" described in Section 4 of [RFC7034]. These document's "site for cookies" is the top-level site if and only if the document and each of its ancestor documents' origins have the same registrable domain as the top-level site. Otherwise its "site for cookies" is the empty string.

Given a Document ("document"), the following algorithm returns its "site for cookies" (either a registrable domain, or the empty string):

1. Let "top-document" be the active document in "document"'s browsing context's top-level browsing context.
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 containing "document" and each of "document"'s ancestor browsing contexts' active documents.
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"'s host's registrable domain is not an exact match for "top-origin"'s host's registrable domain, return the empty string.
5. Return "top-site".

2.1.2. 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 browsing context 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.

2.1.2.1. 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 cookie" values, the worker's "site for cookies" will be the empty string in cases where the values diverge, and the shared value in cases where the values agree.

Given a WorkerGlobalScope ("worker"), the following algorithm returns its "site for cookies" (either a registrable domain, or the empty string):

1. Let "site" be "worker"'s origin's host's registrable domain.
2. For each "document" in "worker"'s Documents:
    1. Let "document-site" be "document"'s "site for cookies" (as defined in Section 2.1.1).
    2. If "document-site" is not an exact match for "site", return the empty string.
3. Return "site".

2.1.2.2. 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.

Requests which simply pass through a service worker will be handled as described above: the request's client will be the Document or Worker which initiated the request, and its "site for cookies" will be those defined in Section 2.1.1 and Section 2.1.2.1

Requests which are initiated by the Service Worker itself (via a direct call to "fetch()", for instance), on the other hand, will have a client which is a ServiceWorkerGlobalScope. Its "site for cookies" will be the registrable domain of the Service Worker's URI.

Given a ServiceWorkerGlobalScope ("worker"), the following algorithm returns its "site for cookies" (either a registrable domain, or the empty string):

1. Return "worker"'s origin's host's registrable domain.

3. Server Requirements

This section describes extensions to [RFC6265] necessary to implement the server-side requirements of the "SameSite" attribute.

3.1. Grammar

Add "SameSite" to the list of accepted attributes in the "Set-Cookie" header field's value by replacing the "cookie-av" token definition in Section 4.1.1 of [RFC6265] with the following ABNF grammar:

```
cookie-av      = expires-av / max-age-av / domain-av /
                 path-av / secure-av / httponly-av /
                 samesite-av / extension-av
samesite-av    = "SameSite" / "SameSite=" samesite-value
samesite-value = "Strict" / "Lax"
```

3.2. Semantics of the "SameSite" Attribute (Non-Normative)

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 2.1. For example, requests for "https://example.com/sekrit-image" will attach same-site cookies if and only if initiated from a context whose "site for cookies" is "example.com".

If the "SameSite" attribute's value is "Strict", or if the value is invalid, 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 4.1.1.

The changes to the "Cookie" header field suggested in Section 4.3 provide additional detail.

4. User Agent Requirements

This section describes extensions to [RFC6265] necessary in order to implement the client-side requirements of the "SameSite" attribute.

4.1. The "SameSite" attribute

The following attribute definition should be considered part of the the "Set-Cookie" algorithm as described in Section 5.2 of [RFC6265]:

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

1. If "cookie-av"'s "attribute-value" is not a case-insensitive match for "Strict" or "Lax", ignore the "cookie-av".
2. Let "enforcement" be "Lax" if "cookie-av"'s "attribute-value" is a case-insensitive match for "Lax", and "Strict" otherwise.
3. Append an attribute to the "cookie-attribute-list" with an "attribute-name" of "SameSite" and an "attribute-value" of "enforcement".

4.1.1. "Strict" and "Lax" enforcement

By default, same-site cookies will not be sent along with top-level navigations. As discussed in Section 5.2, 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 [RFC7231] sense) HTTP method.

Lax enforcement provides reasonable defense in depth against CSRF attacks that rely on unsafe HTTP methods (like "POST"), but do 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 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.

When possible, developers should use a session management mechanism such as that described in Section 5.2 to mitigate the risk of CSRF more completely.

4.2. Monkey-patching the Storage Model

Note: There's got to be a better way to specify this. Until I figure out what that is, monkey-patching!

Alter Section 5.3 of [RFC6265] as follows:

1. Add "samesite-flag" to the list of fields stored for each cookie. This field's value is one of "None", "Strict", or "Lax".
2. Before step 11 of the current algorithm, add the following:

    1. If the "cookie-attribute-list" contains an attribute with an "attribute-name" of "SameSite", set the cookie's "samesite-flag" to "attribute-value" ("Strict" or "Lax"). Otherwise, set the cookie's "samesite-flag" to "None".
    2. If the cookie's "samesite-flag" is not "None", and the request which generated the cookie's client's "site for cookies" is not an exact match for "request-uri"'s host's registrable domain, then abort these steps and ignore the newly created cookie entirely.

4.3. Monkey-patching the "Cookie" header

Note: There's got to be a better way to specify this. Until I figure out what that is, monkey-patching!

Alter Section 5.4 of [RFC6265] as follows:

1. Add the following requirement to the list in step 1:

    * If the cookie's "samesite-flag" is not "None", and the HTTP request is cross-site (as defined in Section 2.1 then exclude the cookie unless all of the following statements hold:

        1. "samesite-flag" is "Lax"
        2. The HTTP request's method is "safe".
        3. The HTTP request's target browsing context is a top-level browsing context.

Note that the modifications suggested here concern themselves only with the "site for cookies" of the request's client, and the registrable domain of the resource being requested. The cookie's "domain", "path", and "secure" attributes do not come into play for these comparisons.

5. Authoring Considerations

5.1. 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.

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.

5.2. 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

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://example.com/". They might expect that clicking on an emailed link to "https://projects.com/secret/project" would show them the secret project that they're authorized to see, but if "projects.com" has marked their session cookies as "SameSite", then this cross-site navigation won't send them along with the request. "projects.com" 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 conceptualy granting "read" access, another granting "write" access. The latter could be marked as "SameSite", and its absence would provide a reauthentication step before executing any non-idempotent action. The former could drop the "SameSite" attribute entirely, or choose the "Lax" version of enforcement, in order to allow users access to data via top-level navigation.

5.3. Mashups and Widgets

The "SameSite" attribute is inappropriate for some important use-cases. In particular, note that content intended for embedding in a cross-site contexts (social networking widgets or commenting services, for instance) will not have access to such cookies. Cross-site cookies may be required in order to provide seamless functionality that relies on a user's state.

Likewise, some forms of Single-Sign-On might require authentication in a cross-site context; these mechanisms will not function as intended with same-site cookies.

6. Privacy Considerations

6.1. Server-controlled

Same-site cookies in and of themselves don't do anything to address the general privacy concerns outlined in Section 7.1 of [RFC6265]. The 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. Connection and/or socket pooling, Token Binding, and Channel ID all offer explicit methods of identification that servers could take advantage of.

6.2. Pervasive Monitoring

As outlined in [RFC7258], pervasive monitoring is an attack. Cookies play a large part in enabling such monitoring, as they are responsible for maintaining state in HTTP connections. We considered restricting same-site cookies to secure contexts [secure-contexts] as a mitigation but decided against doing so, as this feature should result in a strict reduction in the number of cookies floating around in cross-site contexts. That is, even if "http://not-example.com" embeds a resource from "http://example.com/", that resource will not be "same-site", and "http://example.com"'s cookies simply cannot be used to correlate user behavior across distinct origins.

7. References

7.1. Normative References

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

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

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

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

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

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

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

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

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

[SERVICE-WORKERS] Russell, A., Song, J., and J. Archibald, "Service Workers", n.d., <http://www.w3.org/TR/service-workers/>.

7.2. Informative References

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

[pixel-perfect] Stone, P., "Pixel Perfect Timing Attacks with HTML5", n.d., <http://www.contextis.com/documents/2/Browser_Timing_Attacks.pdf>.

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

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

[samedomain-cookies] Goodwin, M. and J. Walker, "SameDomain Cookie Flag", 2011, <http://people.mozilla.org/~mgoodwin/SameDomain/samedomain-latest.txt>.

[secure-contexts] West, M., "Secure Contexts", n.d., <https://w3c.github.io/webappsec-secure-contexts/>.

Appendix A. Acknowledgements

The same-site cookie concept documented here is indebited to Mark Goodwin's and Joe Walker's [samedomain-cookies]. Michal Zalewski, Artur Janc, Ryan Sleevi, and Adam Barth provided particularly valuable feedback on this document.

Authors' Addresses

Mike West
Google, Inc

Email: mkwst@google.com
URI: https://mikewest.org/

Mark Goodwin
Mozilla

Email: mgoodwin@mozilla.com
URI: https://www.computerist.org/