Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 16, 2020

M. West
Google

March 15, 2020

Incrementally Better Cookies
draft-west-cookie-incrementalism-01

Abstract

This document proposes a few changes to cookies inspired by the properties of the HTTP State Tokens mechanism proposed in [I-D.west-http-state-tokens]. First, cookies should be treated as "SameSite=Lax" by default. Second, cookies that explicitly assert "SameSite=None" in order to enable cross-site delivery should also be marked as "Secure". Third, same-site should take the scheme of the sites into account. Fourth, cookies should respect schemes. Fifth, cookies associated with non-secure schemes should be removed at the end of a user's session. Sixth, the definition of a session should be tightened.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The HTTP State Tokens proposal ([I-D.west-http-state-tokens]) aims to replace cookies with a state management mechanism that has better security and privacy properties. That proposal is somewhat aspirational: it's going to take a long time to come to agreement on the exact contours of a cookie replacement, and an even longer time to actually do so.

While we're debating the details of a new state management primitive, it seems quite reasonable to reevaluate some aspects of the existing primitive: cookies. When we can find consensus on some aspect of HTTP State Tokens, we can apply those aspirations to cookies, driving incremental improvements to state management in the status quo.

Based on conversations at [HTTP-Workshop-2019] and elsewhere, I'd suggest that we have something like agreement on at least three principles:

1. HTTP requests should not carry state along with cross-site requests by default (see Section 8.2 of [RFC6265bis]).

2. HTTP requests should not carry state over non-secure channels (see Section 8.3 of [RFC6265bis], and [RFC7258]).

3. Non-secure channels should not be able to infuence the state of securely-transported content (see Sections 8.3, 8.5, and 8.6 of [RFC6265bis]).

With those principles in mind, this document proposes a few changes that seem possible to deploy in the near-term. User agents should:

1. Treat the lack of an explicit "SameSite" attribute as "SameSite=Lax". That is, the "Set-Cookie" value "key=value" will produce a cookie equivalent to "key=value; SameSite=Lax". Cookies that require cross-site delivery can explicitly opt-into such behavior by asserting "SameSite=None" when creating a cookie.

   This is spelled out in more detail in Section 3.1.

2. Require the "Secure" attribute to be set for any cookie which asserts "SameSite=None" (similar conceptually to the behavior for the "__Secure-" prefix). That is, the "Set-Cookie" value "key=value; SameSite=None; Secure" will be accepted, while "key=value; SameSite=None" will be rejected.

   This is spelled out in more detail in Section 3.2.

3. Require both the scheme and registrable domain of a request's client's "site for cookies" to match the target URL when deciding whether a given request is considered same-site. That is, a request initiated from "http://site.example" to "https://site.example" should be considered cross-site.

   This is spelled out in more detail in Section 3.3.

4. Separate cookies by scheme. That is, a given cookie set from "http://example.com/" should be considered distinct from the same cookie set from "https://example.com/", preventing the former from influencing the state of the latter.

   This is spelled out in more detail in Section 3.4.

5. Evict non-secure cookies when a user's session on a non-secure site ends, thereby reducing the timespan over which a user broadcasts a stable identifier to the network.

   This is spelled out in more detail in Section 3.5.

6. Tighten the definition of a user's "session" with heuristics that better represent users' expectations.

   This is spelled out in more detail in Section 3.6.

2. Conventions and Definitions

2.1. Conformance

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.

2.2. Syntax

This document adjusts some syntax from [RFC6265bis], and in doing so, relies upon the Augmented Backus-Naur Form (ABNF) notation of [RFC5234].

3. Monkey-Patches against RFC6265bis

3.1. "Lax" by Default

The processing algorithm in Section 5.3.7 of [RFC6265bis] treats the absence of a "SameSite" attribute in a "Set-Cookie" header as equivalent to the presence of "SameSite=None". Cookies are therefore available for cross-site delivery by default, and developers may opt-into more security by setting some other value explicitly. Ideally, we'd invert that such that developers who accepted the risks of cross-site delivery (see Section 8.2 of [RFC6265bis]) could opt into them, while developers who didn't make any explicit choice would be protected by default.

We could accomplish this goal by first altering the processing algorithm, replacing the current step 1:

1. Let "enforcement" be "None".

with the following two steps:

1. Let "enforcement" be "Default".

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

And then by, altering step 13 of the cookie storage model (Section 5.4 of [RFC6265bis]) from:

13. If the cookie-attribute-list contains an attribute with an attribute-name of "SameSite", set the cookie's same-site-flag to attribute-value (i.e. either "Strict", "Lax", or "None"). Otherwise, set the cookie's same-site-flag to "None".

to:

13. 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 attribute-value. Otherwise, set the cookie's same-site-flag to "Default".

And finally by altering the fifth bullet point of step 1 of the cookie-string construction algorithm in Section 5.5 of [RFC6265bis] from:

* If the cookie's same-site-flag is not "None", and the HTTP request is cross-site (as defined in Section 5.2) then exclude the cookie unless all of the following statements hold:

  1. The same-site-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.

to:

- If the cookie's same-site-flag is not "None", and the HTTP request is cross-site (as defined in Section 5.2) then exclude the cookie unless all of the following statements hold:

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

   2. The HTTP request's method is "safe".

   3. The HTTP request's target browsing context is a top-level browsing context.

This would have the effect of mapping the default behavior in the absence of an explicit "SameSite" attribute, as well as the presence of any unknown "SameSite" value, to the "Lax" behavior, protecting developers by making cross-site delivery an explicit choice, as opposed to an implicit default.

3.1.1. "Lax-Allowing-Unsafe" Enforcement

The "Lax" enforcement mode described in Section 5.3.7.1 of [RFC6265bis] allows a cookie to be sent along with cross-site requests if and only if they are top-level navigations with a "safe" HTTP method. Implementation experience shows that this is difficult to apply across the board, and it may be reasonable to temporarily carve out cases in which some cookies that rely on today's default behavior can continue to be delivered as the default is shifted to "Lax" enforcement.

One such carveout, described in this section, accommodates certain cases in which it may be desirable for a cookie to be excluded from non-top-level cross-site requests, but to be sent with all top-level navigations regardless of HTTP request method.

For example, a login flow may involve a cross-site top-level POST request to an endpoint which expects a cookie with login information. 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. On the other hand, "None" enforcement would allow the cookie to be sent with all cross-site requests. For a cookie containing potentially sensitive login information, this may not be desirable.

In order to retain some of the protections of "Lax" enforcement (as compared to "None") while still allowing cookies to be sent cross-site with unsafe top-level requests, user agents may choose to provide an intermediate "Lax-allowing-unsafe" enforcement mode. A cookie whose enforcement mode is "Lax-allowing-unsafe" will be sent along with a cross-site request if and only if it is a top-level request, regardless of request method.

User agents may choose to apply this enforcement mode instead of "Lax" enforcement, but only in a limited or restricted fashion. Such restrictions may include applying "Lax-allowing-unsafe" only to cookies that did not explicitly specify "SameSite=Lax" (i.e., those whose same-site-flag was set to "Default" by default) with creation-time more recent than a duration of the user agent's choosing (2 minutes seems reasonable).

This is done by further modifying the previously mentioned fifth bullet point of step 1 of the cookie-string construction algorithm in Section 5.5 of [RFC6265bis] from:

* If the cookie's same-site-flag is not "None", and the HTTP request is cross-site (as defined in Section 5.2) then exclude the cookie unless all of the following statements hold:

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

  2. The HTTP request's method is "safe".

  3. The HTTP request's target browsing context is a top-level browsing context.

to:

* If the cookie's same-site-flag is not "None", and the HTTP request is cross-site (as defined in Section 5.2) then exclude the cookie unless all of the following statements hold:

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

  2. The HTTP request's method is "safe", or the cookie meets the user agent's requirements for being granted "Lax-allowing-unsafe" enforcement.

  3. The HTTP request's target browsing context is a top-level browsing context.

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 measure to ease adoption of "Lax" enforcement by default.

3.2. Requiring "Secure" for "SameSite=None"

Cookies sent over plaintext HTTP are visible to anyone on the network. As section 8.3 of [RFC6265bis] points out, this visibility exposes substantial amounts of data to network attackers. We know, for example, that long-lived and stable cookies have enabled pervasive monitoring [RFC7258] in the past (see Google's PREF cookie [pref-cookie]), and we know that a secure transport layer provides significant confidentiality protections against this kind of attack.

We can, to a reasonable extent, mitigate this threat by ensuring that cookies intended for cross-site delivery (and therefore likely to be more prevalent on the wire than cookies scoped down to same-site requests) require secure transport.

That is, we can require that any cookie which asserts "SameSite=None" must also assert the "Secure" attribute (Section 4.1.2.5 of [RFC6265bis]) by altering the storage model defined in Section 5.4 of [RFC6265bis], inserting the following step after the existing step 14:

15. 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.

This is conceptually similar to the requirements put into place for the "__Secure-" prefix (Section 4.1.3.1 of [RFC6265bis]).

3.3. Schemeful Same-Site

By considering the scheme as well as the registrable domain when determining whether a given request is "same-site", the "SameSite" attribute can protect secure origins from CSRF attacks initiated by a network attacker that can forge requests from a non-secure origin on the same registrable domain. To do so we need to modify a number of things:

First change the definition of "site for cookies" from a registrable domain to an origin. In the places where a we return an empty string for a non-existent "site for cookies" we should instead return an origin set to a freshly generated globally unique identifier. Then replace the same-site calculation algorithm with the following:

Two origins, A and B, are considered same-site if the following algorithm returns true:

```
1.  If A and B are both scheme/host/port triples then

    1.  If A's scheme does not equal B's scheme, return false.

    2.  Let hostA be A's host, and hostB be B's host.

    3.  If hostA equals hostB and hostA's registrable domain is null, return
true.

    4.  If hostA's registrable domain equals hostB's registrable domain and is
non-null, return true.

2.  If A and B are both the same globally unique identifier, return true.

3.  Return false.
```

Note: The port component of the origins is not considered.

A request is "same-site" if its target's URI's origin is same-site with the request's client's "site for cookies", or if the request has no client. The request is otherwise "cross-site".

Now that we have a new algorithm, we can update any comparision of two sites from "have the same registrable domain" (or "is an exact match for") to say "is same-site".

Note: The request's URL when establishing a WebSockets connection has scheme "http" or "https", rather than "ws" or "wss". FETCH maps schemes when constructing the request. This mapping allows same-site cookies to be sent with WebSockets.

3.4. Scheming Cookies

Cookies are one of the very few components of the web platform that ignore scheme by default. The "Secure" attribute can lock a cookie to secure schemes, and the "__Secure-" prefix can harden that boundary, but these mechanisms are little-used, and cookies lacking these protections flow across scheme boundaries. They are delivered to both the HTTP and HTTPS variants of a given domain, even though their security properties differ radically. As Section 8.6 of [RFC6265bis] points out, this gives network attackers the ability to influence otherwise secured traffic by modifying user state that flows to secure origins, and, of course, insight into user behavior as securely-set cookies that lack the "Secure" attribute likewise flow from secure origins to non-secure variants.

We should remedy this defect by storing a "scheme" component along with the cookie, and using that component in cookies' matching algorithms to ensure that secure and non-secure origins' state is clearly distinguishable and separate. This is accomplished as follows:

First, alter the Storage Model defined in Section 5.4 of [RFC6265bis] by adding "scheme" to the list of fields the user agent stores about each cookie, and setting it when creating a cookie by altering step 2 of the same algorithm from:

2. 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.

to:

2. 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. Set the scheme to request-uri's origin's scheme component.

Likewise alter step 17 of the same algorithm from:

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

to:

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

And step 17.1 from:

17.1. 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.)

to:

17.1. **Let old-cookie be the existing cookie with the same name, scheme, 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.)

Second, alter The Cookie Header algorithm defined in Section 5.5 of [RFC6265bis] to take the "scheme" into account when deciding which cookies to deliver by adding another condition to the list in Step 1 of the algorithm:

* The cookies' `scheme` matches the scheme component of request-uri's origin.

This seems like the minimal set of changes necessary. We could do other cleanup, including removing the "Secure" attribute, as this mechanism obviates it entirely, altering the eviction algorithm to prefer discarding non-secure schemes, etc.

3.5. Evict Non-Secure Cookies

In the status quo, cookies delivered to non-secure origins are, generally, quite old. Each cookies' age is somewhat representative of its risk: long-lived cookies expose persistent identifiers to the network when delivered non-securely which create tracking opportunities over time. Here, we aim to mitigate this risk by substantially reducing the lifetime of non-secure cookies, thereby limiting the window of opportunity for network attackers.

This is similar conceptually to previous proposals, notably [I-D.thomson-http-omnomnom] and [cookies-over-http-bad], but seems like it might be more deployable, especially in conjunction with the scheme changes above.

The change is straightforward, requiring the following text to be added to the bottom of Section 5.4 of [RFC6265bis]:

~~~ When "the current session is over", the user agent MUST remove from the cookie store all cookies whose "scheme" component is non-secure. ~~

As discussed below in {#session-lifetime}, if we add a site-specific session concept, we can make the following addition:

~~ When "the current session is over" for an origin, the user agent MUST remove from the cookie store all cookies whose "scheme" component is non-secure, and whose "domain" component's registrable domain matches the origin's registrable domain. ~~

This still requires the user agent to define a notion of non-secureness, but it would certainly include "http".

3.6. Session Lifetime

Section 5.4 of [RFC6265bis] defines "the current session is over" by choosing not to define it, instead leaving it up to the user agent. Unfortunately, we have several "session" concepts in user agents today, and it's not clear that any of them are appropriate for cookies. HTML's "sessionStorage" lifetime is tied to a particular top-level browsing context, thereby giving two tabs/windows different
~~~

views into a page's state. Various user agents' "private mode" create sessions that are scoped in various ways: Chrome's Incognito mode ties a session's lifetime to the closure of the last Incognito window, Safari's private mode's lifetime is tab-specific, etc. Session cookies' lifetime likewise differs between user agents, in some cases based on user-visible settings like Chrome's "Continue where you left off" (which can lead to quite persistent sessions indeed).

At some risk of further complicating the notion of a "session", it might be reasonable to learn from existing user agents' work around meeting users' conceptions of when they're using a given site, and to define a recommended heuristic that user agents could adopt. In particular, Chromium's site engagement score and Safari's ITP both track a user's last moment of interaction with a site (which might feasibly include things like navigation, clicks, scrolls, etc). This seems like a useful bit of data to take into account, along with whether or not a user has top-level browsing contexts that include a given site.

To that end, we could add a few concepts to [RFC6265bis] to give browser vendors more clarity around a reasonable approach to defining when "the current session is over" for a specific site, rather that for the browsing session as a whole. Something along the following lines makes sense to me:

1. User agents should store a timestamp of the last interaction with a given site in a top-level browsing context [HTML]. User agents have a great deal of flexibility in what they consider an interaction, but typing and clicking should probably count.

2. Change the "close a browsing context" algorithm [HTML] to call the following algorithm between its existing step 1 and step 2:

    1. Let "closedOrigin" be the origin of "browsingContext"'s active document.

    2. For each top-level browsing context "c":

        1. If "c" is "browsingContext", continue.

        2. If "c"'s active document's origin is same site with "browsingContext"'s active document's origin, return.

    3. ASSERT: No top-level browsing context contains a document that's same-site with the document being closed.

    4. Return, and continue running this algorithm in parallel.

```
    5.  Wait however long a user would reasonably expect their state
        to be retained (an hour sounds reasonable).

    6.  For each top-level browsing context "c":

        1.  If "c"'s active document's origin is same site with
            "closedOrigin", return.

    7.  ASSERT: No top-level browsing context contains a document
        that's same-site with the document that was closed.

    8.  Trigger "the current session is over" for "closedOrigin".

3.  Define a new handler for "the current session is over" that takes
    an origin into account, and clears session cookies for that
    origin's site.
```

Note that these definitions refer to "site", not "origin", as cookies span an entire registrable domain. Ideally, we'll address that too, but not today.

4. Security and Privacy Considerations

4.1. CSRF

"SameSite" is a reasonably robust defense against some classes of cross-site request forgery attacks, as described in Section 8.8.1 of [RFC6265bis], but developers need to opt-into its protections in order for them to have any effect. That is, developers are vulnerable to CSRF attacks by default, and must do some work to shift themselves into a more defensible position.

The change proposed in Section 3.1 would invert that requirement, placing the burden on the small number of developers who are building services that require state in cross-site requests. Those developers would be empowered to opt-into the status quo's less-secure model, while developers who don't intend for their projects to be embedded in cross-site contexts are protected by default.

4.2. Secure Transport

As discussed in Section 8.3 of [RFC6265bis], cookies delivered over plaintext channels are exposed to intermediaries, and thereby enable pervasive monitoring [RFC7258]. The change proposed in Section 3.2 above would set secure transport as a baseline requirement for all stateful cross-site requests, thereby reducing the risk that these cookies can be cataloged or modified by network attackers.

Requiring secure transport for cookies intended for cross-site usage has the exciting secondary effect of increasing pressure on entities that produce embeddable content to migrate their products to HTTPS. That has security benefits for those third-party products themselves, but also has the effect of removing the potential of mixed content ([mixed-content]) as a blocker to first-party migration to HTTPS.

Note that in the long term, it seems quite reasonable to take the additional step of requiring the "Secure" attribute for all cookies, regardless of their "SameSite" value. That would have more substantial impact on pervasive monitoring and network attackers generally. This document's proposal limits itself to "SameSite=None" because that seems like a low-hanging, high-value change that's deployable in the near term. User agents are encouraged to find additional subsets for which "Secure" can be required.

4.3. Tracking

The proposals in this document do not in themselves mitigate the privacy risks described in Section 7.1 of [RFC6265bis]. Entities who wish to use cookies to track user activity from cross-site contexts can continue to do so by setting cookies that declare themselves as "SameSite=None".

Requiring that explicit declaration, however, gives user agents the ability to easily distinguish cookies used for stateful cross-site requests from those with narrower scope. After the change proposed in Section 3.1, only those cookies that make an explicit "SameSite=None" declaration can be directly used for cross-site tracking. It may make sense for user agents to use that information to give users different controls for these cookies, or to apply different policies for expiration and delivery.

5. Implementation Considerations

5.1. Sequencing

The steps described in this document don't need to be taken at the same time. It's quite possible that it will be less disruptive to deploy "SameSite=Lax" as a default first, then to require the "Secure" attribute for any explicitly "SameSite=None" cookie as a subsequent step, and then deploying schemeful same-site in a final step.

User agents are encouraged to adopt these recommendations in whatever order they believe will lead to the widest, most expedient deployment.

5.2. Deployment

It's possible that a middle-ground between "SameSite=Lax" and "SameSite=None" could be a better balance between doing what developers want by default, and mitigating CSRF by default. [I-D.west-cookie-samesite-firstparty] explores the possibility of integrating First-Party Sets [first-party-set] with the "SameSite" attribute in order to allow entities that shard themselves across multiple registrable domains to maintain stateful communication between them (to support single-sign on, for example).

It's possible that user agents who support First-Party Sets could reduce the deployment overhead for developers, and increase the robustness of a site's CSRF defense for cross-site-but-not-cross-party cookies by defaulting to something like that document's "FirstPartyLax" instead of "Lax".

6. IANA Considerations

This document has no IANA actions.

7. References

7.1. Normative References

[HTML] "HTML", n.d., <https://html.spec.whatwg.org/>.

[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/info/rfc2119>.

[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/info/rfc5234>.

[RFC6265bis] West, M. and J. Wilander, "Cookies: HTTP State Management Mechanism", draft-ietf-httpbis-rfc6265bis-05 (work in progress), February 2020.

[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/info/rfc8174>.

7.2. Informative References

[cookies-over-http-bad]
West, M., "Cookies over HTTP Bad", April 2018, <https://github.com/mikewest/cookies-over-http-bad>.

[first-party-set]
West, M., "First-Party Sets", n.d., <https://mikewest.github.io/first-party-sets/>.

[HTTP-Workshop-2019]
Nottingham, M., "HTTP Workshop 2019: Report", April 2019, <https://github.com/HTTPWorkshop/workshop2019/wiki/Report>.

[I-D.thomson-http-omnomnom]
Thomson, M. and C. Peterson, "Expiring Aggressively Those HTTP Cookies", draft-thomson-http-omnomnom-00 (work in progress), May 2016.

[I-D.west-cookie-samesite-firstparty]
West, M., "First-Party Sets and SameSite Cookies", draft-west-cookie-samesite-firstparty-01 (work in progress), May 2019.

[I-D.west-http-state-tokens]
West, M., "HTTP State Tokens", draft-west-http-state-tokens-00 (work in progress), March 2019.

[mixed-content]
West, M., "Mixed Content", n.d., <https://w3c.github.io/webappsec-mixed-content/>.

[pref-cookie]
Soltani, A., Peterson, A., and B. Gellman, "NSA uses Google cookies to pinpoint targets for hacking", December 2013, <https://www.washingtonpost.com/news/the-switch/wp/2013/12/10/nsa-uses-google-cookies-to-pinpoint-targets-for-hacking/>.

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

Acknowledgments

Conversations with a number of folks at 2019's HTTP Workshop helped me clarify my thinking around the incremental improvements we can make to cookies. In particular, Martin Thomson and Anne van Kesteren provided insightful feedback.

Lily Chen has been instrumental in initial deployments of the "SameSite" changes described in Section 3.1 and Section 3.2, proving that incremental changes to cookies can be successfully shipped.

Steven Bingler contributed the "Schemeful SameSite" proposal described in Section 3.3.

Author's Address

Mike West
Google

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