The Future of Cookies 🍪

…in Web Standards

This session is part of an ongoing conversation in Privacy CG, IETF HTTP WG, and more.

We won’t find all the solutions today, so let’s keep discussing it.

Background: Specifications

- RFC 6265(bis) is the cookie standard (in the IETF HTTP WG).
- Specifies cookies and cookie handling for browsers and other clients.
- Browsers have additional features:
    - document.cookie (“non-HTTP” API”)
    - <iframe>s and subresources
- Fetch/HTML describe when Cookie headers are sent and when Set-Cookie headers are parsed, but integration with the RFC is not ideal:
    - document.cookie vs. network request cookie store updates unclear.
    - In implementations, the network layer has no access to the DOM and can’t actually implement the RFC.

Cookie Innovation in Browsers

- Increased focus on security and privacy means all browsers want to restrict (cross-site) cookies in new ways.
- Some innovations may benefit other clients on the network layer, e.g.:
  - Origin-bound Cookies by Default
  - Expiring Aggressively Those HTTP Cookies
- However, browsers also increasingly innovate on cookies without relevance to other clients.
- It’s difficult to fully specify these new APIs and restrictions in RFC 6265bis alone.

SameSite Cookies

- The *SameSite* attribute may be used to enforce or loosen restrictions on cookies in nested cross-site documents.
- Used as a security measure against, e.g., CSRF, not generally for privacy.
- Values are
  - None: Cookie is sent in a cross-site context
  - Lax: Cookie is not sent in a cross-site context but in cross-site top-level navigations.
  - Strict: Cookie is not sent in cross-site context or cross-site top-level navigations.
- None is default in Safari, Firefox[1]. Chrome default is “Lax”.

[1] Firefox is attempting to ship “Lax” by default but runs into compat issues with Chrome, see https://github.com/httpwg/http-extensions/issues/2104

SameSite Cookies

Challenges:

- **“Site for cookies” needs to take document nesting into account, which isn’t known on the network layer.**
- **RFC should not concern itself with Workers, Service Workers, etc.**

Partitioned Cookies (CHIPS)

- The *Partitioned* attribute can be used to expose cookies in nested cross-site documents, keyed by their respective top-level.
- This can maintain website functionality when cross-site cookies are otherwise blocked.
- Partitioning also exists for Storage and Network (Cache)

Challenges:

- **Cookies, Storage, Network all have different partition keys.**
- **Network layer can’t determine when/how to partition.**

Blocking cross-site cookies & SAA

- Browsers have announced plans to restrict (mostly block) cross-site cookies by default.
- Exceptions can be granted through the Storage Access API (SAA) or by using partitioned cookies.
- Some browsers have more complicated rules, such as ephemeral cookies.

Challenges:

- **How is “cross-site” defined (next discussion)?**
- **Network layer can’t access embedding structure or storage access exceptions.**

Discussion Topics

- **Cookie Layering**
- **Third-party cookie blocking semantics**
- **Other Open Questions (might not have time):**
  - Can partition keys be consistent between storage and cookies (and network)?
  - Is there a need for a browser signal when a site is in an embedded/partitioned context?
  - Do we need to specify some concept of “ephemeral” cookies, i.e. additional expiry types?
- **…?**

Cookie Layering

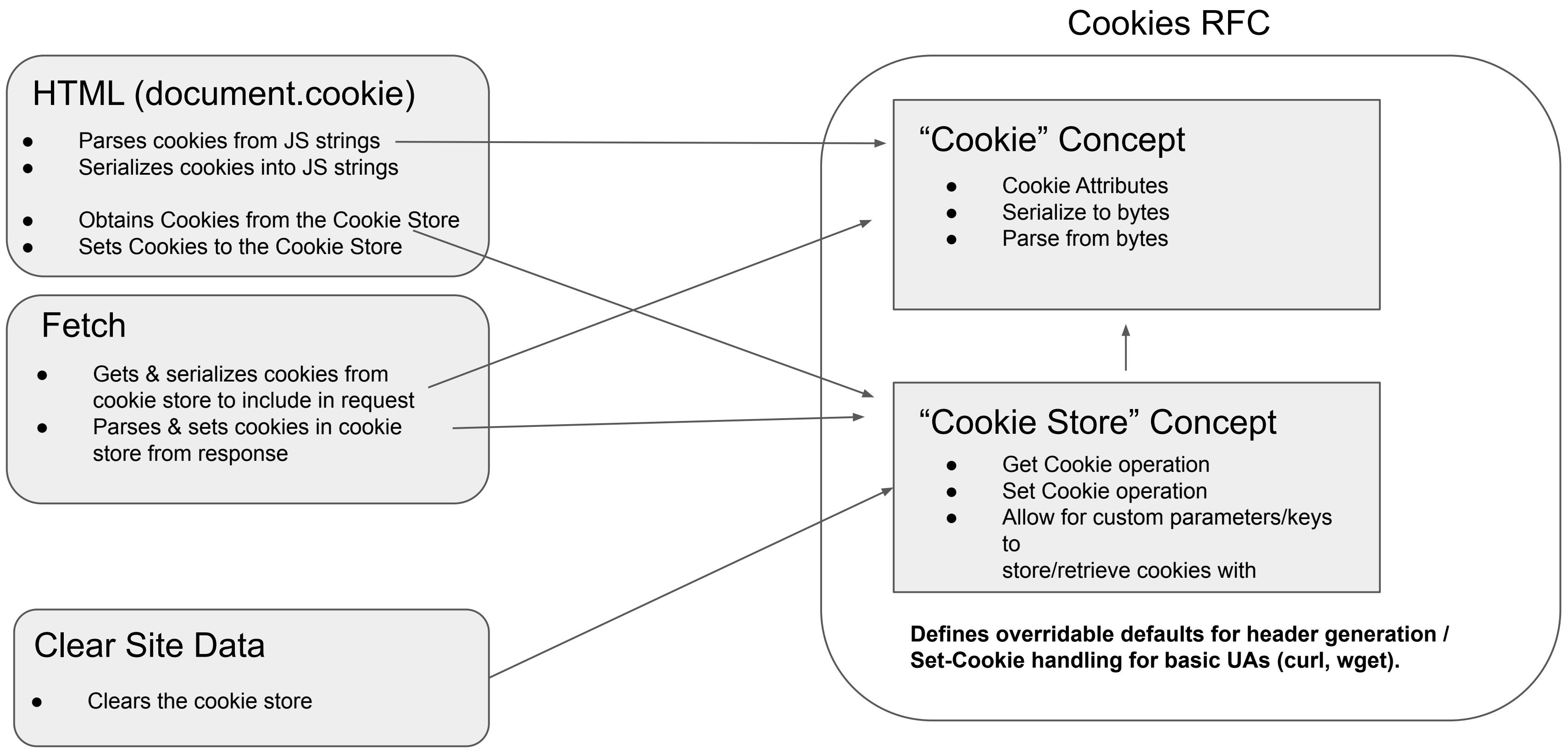
Cookie Layering

- Layering helps us separate concerns by abstracting shared logic in a common base layer (the Cookies RFC), while keeping browser-focused logic in separate, dependent specifications (HTML, Fetch, etc.)
- Proposed in Cookie layering #2084 - httpwg/http-extensions

Goals for a new architecture?

- Share common base logic
    - **Discuss: What does this mean for us?**
    - E.g., keep a single grammar, same parser, same serializer, same model, share all attributes, including SameSite, Partitioned etc.
- Allow browsers to specify more complicated rules for storing and retrieving cookies as part of HTTP and non-HTTP APIs.
    - **Discuss: Should web standards exclusively specify SameSite behavior, Partitioned, 3PC blocking, etc.? Can we move all browser-specific pieces out of the RFC?**
        - Workers
        - SameSite mechanics
        - 3P cookie handling
        - Partitioned cookies
    - **Discuss: Should we allow browsers to store additional information in the cookie store?**
        - Top-level site for partitioned
        - Future: Whether cookie was set by non-HTTP, CNAME’d etc.

Rough Idea: Division of labor



Cookies RFC

HTML (document.cookie)

- Parses cookies from JS strings
- Serializes cookies into JS strings

- Obtains Cookies from the Cookie Store
- Sets Cookies to the Cookie Store

Fetch

- Gets & serializes cookies from cookie store to include in request
- Parses & sets cookies in cookie store from response

Clear Site Data

- Clears the cookie store

“Cookie” Concept

- Cookie Attributes
- Serialize to bytes
- Parse from bytes

“Cookie Store” Concept

- Get Cookie operation
- Set Cookie operation
- Allow for custom parameters/keys to store/retrieve cookies with

Defines overridable defaults for header generation / Set-Cookie handling for basic UAs (curl, wget).

Division of labor

IETF:

- "Cookie" concept
- "Cookie" to bytes (serializer) for the Cookie (and Set-Cookie for servers) header
- Bytes to "Cookie" (parser) for the Set-Cookie (and Cookie for servers) header
- "Cookie store" concept, including get and set operations (with suitable parameters) which return and accept "Cookie" concepts respectively.
    - This will have default handling for all "Cookie" attributes, such as SameSite. Not all user agents are expected to pass in different values, but the operations will be able to handle other values. How those other values are computed is not necessarily part of the standard.
    - Idea: have a generic key field that other specifications (see below) can fill in upon which normal equality operations are used when get and set.
- Does not (at least not for all UAs) define Cookie header generation for requests and Set-Cookie handling for responses. Does provide defaults for both that should be suitable for general UAs such as curl and wget.

WHATWG Fetch & HTML

- Defines when and how (key) cookies are obtained from the "Cookie store" and included in a request.
- Defines when and how (key) cookies are parsed and then set in the "Cookie store" for a response.
- Defines how document.cookie interacts with the "Cookie store".
- Define clearing operations for the store, used by browsers & Clear-Site-Data
- (Takes care of computation of various parameters for the get and set operations of the "Cookie store", such as SameSite and Partitioned. This might be done early in the fetch algorithm or before fetch is even invoked, depending. But mostly has to happen before fetch goes in parallel as the state is not available at that point.)

Align on semantics of cross-site cookie blocking

Align on semantics of cross-site cookie blocking

- Should SameSite=None cookies continue to be supported even when cross-site cookies are blocked?

- As a reminder:
  - SameSite is intended as a CSRF protection.
  - Chromium (and soon, Gecko) treats Lax[1] as the default SameSite value. Which means that developers must opt-in to make the cookie accessible in cross-site contexts

[1] When SameSite is unspecified, Chromium currently uses "Lax-Allowing-Unsafe" enforcement to account for some site compatibility issues.

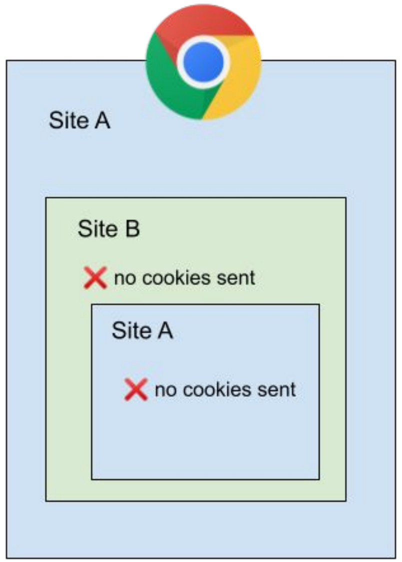
Specific scenarios that need alignment

Scenario 1: Same-site with top-level document, but has one or more cross-site ancestors

- Should we allow (SameSite=None) cookies to be sent in the inner-most frame, since it is same-site with the top-most frame?
- Are there any tracking risks in doing so?
- There appear to be compatibility risks with not allowing this scenario to function, such as with some identity use-cases.
- Current browser behaviors:
  - Chrome blocks cookies set by inner frame “A”.
  - In Firefox, it appears that the inner “A” frame can read cookies, but not set them. Is this intentional?
  - Safari appears to allow cookies to be set by inner “A” frame.

Options:

- Allow inner “A” frame to receive the same set of SameSite=None cookies as the top-level “A” frame
- Only allow inner “A” frame to only set/receive cookies with an additional attribute such as `Partitioned`.



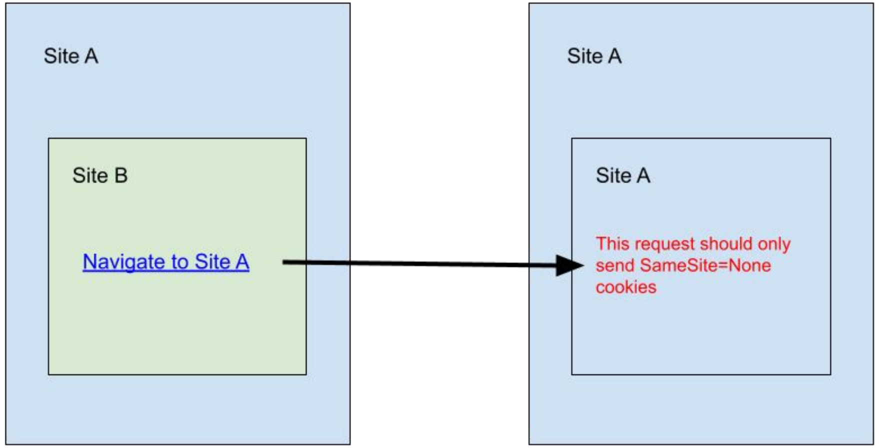
Specific scenarios that need alignment

Scenario 2: Cross-site to same-site embedded requests

- Should we allow (SameSite=None) cookies to be sent on the cross-site request from B to A, since A same-site with the top-most frame?
- As before, there appear to be some use-cases, such as identity-as-a-service, that use this pattern.
- Current browser behaviors:
  - All major browsers appear to allow cookies based on supported SameSite semantics (e.g. Safari allows SameSite-unspecified cookies, while Chrome doesn't)

Options:

- Allow inner "A" frame to receive the same set of SameSite=None cookies as the top-level "A" frame
- Only allow inner "A" frame to only set/receive cookies with an additional attribute such as `Partitioned`.



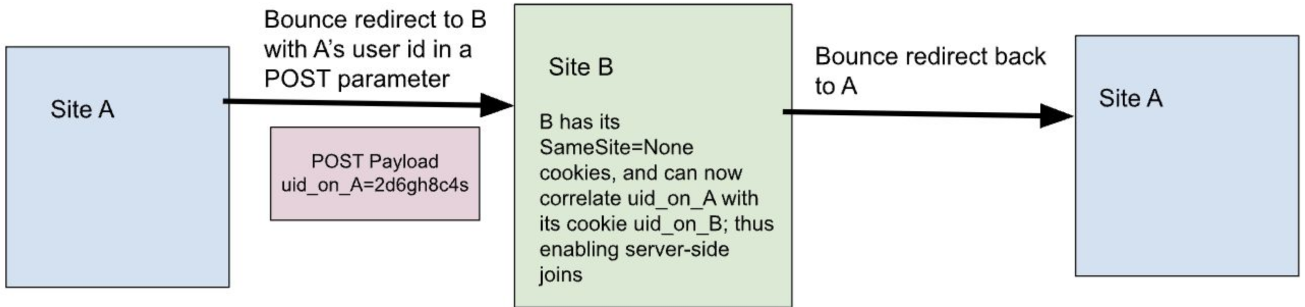
Specific scenarios that need alignment

Scenario 3: Top-level cross-site POST requests

- Should we allow (SameSite=None|Lax-allowing-Unsafe) cookies to be sent unhindered on top-level cross-site POST requests?
- Navigational tracking mitigation work sometimes intervenes by stripping URL parameters
  - Could POST parameters be used as a workaround?
- There appear to be compatibility risks with not allowing this scenario to function, such as with some identity and payments use-cases.

Options:

- Allow SameSite=None cookies on top-level cross-site POSTs, but rely on navigation tracking mitigations to intervene on wide-scale tracking.
- Require these use-cases to move to purpose-built solutions over a period of time.


Site A
Bounce redirect to B with A's user id in a POST parameter
POST Payload uid_on_A=2d6gh8c4s
Site B
B has its SameSite=None cookies, and can now correlate uid_on_A with its cookie uid_on_B; thus enabling server-side joins
Bounce redirect back to A
Site A

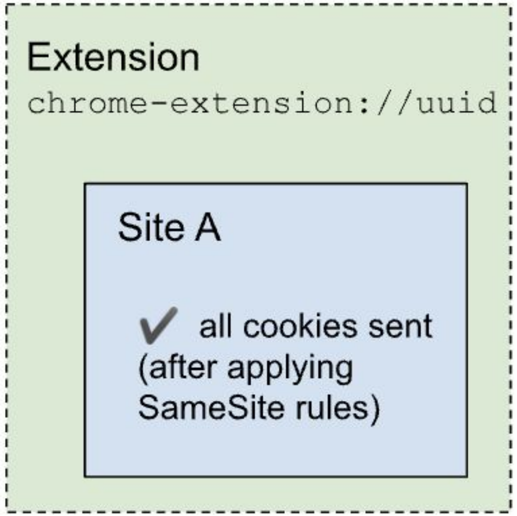
Specific scenarios that need alignment

Scenario 4: Extensions (is this out of scope for W3C?)

- Requests initiated by Chrome extensions are made from a chrome-extension:// URL, which is always cross-site to a http:// or https:// URL.
- Chrome currently allows cross-site requests made from inside extension pages to get cookies even when third-party cookies are blocked (unsure of history).
- [TODO] What is the behavior on other browsers?

Options:

- Continue existing behavior
- Require extension to have host permissions for site A
  - Has undesirable consequences for extensions like PDF viewer in Chrome which now has to request host permissions for "*"
- ???



Current behavior in Chrome when third-party cookies are blocked

Open Discussion