HTTPbis
Internet-Draft
Updates: 6265 (if approved)
Intended status: Standards Track
Expires: December 8, 2017

M. West
Google, Inc

June 6, 2017

Non-HTTP Cookies
draft-west-nonhttp-cookies-00

Abstract

This document updates RFC6265 by defining a "NonHttp" attribute which ensures that a given cookie is available only to "non-HTTP" APIs. Yes, it is a little strange for "HTTP State Management" to support exclusively non-HTTP use cases, but the internet is a strange place. So it goes.

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 8, 2017.

Copyright Notice

Table of Contents

1. Introduction

Cookies violate the same-origin policy in well-understood ways, outlined in Sections 8.5 and 8.6 of [RFC6265]. For better or worse, developers rely on this behavior. In particular, it is quite common for developers to use cookies as a cross-origin storage mechanism, allowing state to be shared across all origins in a given registrable domain. A user who signs into "https://m.example.com/" might expect to also be signed into "https://www.example.com/", and vice-versa. Servers can support this expectation by setting a cookie at the apex domain: "uid=1234567;Secure;Domain=example.com". The cookie will be sent along with HTTP requests to both origins, and the user's state can be maintained server-side.

Often, though, the fact that a cookie is sent over the wire is an unindended side-effect. Developers may be using cookies to easily share client-side state between related origins, and have pressed cookies into this service due to their unique security properties.

For example, web analytics packages might use a user agent's "document.cookie" API to set first-party cookies on a client's site that retain a user's state. This gives them the information they need to do analytics work, but has the very unfortunate side-effect of causing those cookies to be sent along with requests to the first-party's server. This has a substantially negative impact on request size (and therefore performance), and also on privacy, as these

cookies often contain identifiers, and are often sent in plaintext over the network.

This document aims to reduce both impacts by allowing developers to specify specific cookies as "NonHttp". These cookies cannot be set or modified via "Set-Cookie" HTTP response headers, nor will they be included in "Cookie" HTTP request headers. They are only accessible via "non-HTTP" APIs, satisfying purely client-side storage requirements.

1.1. Examples

Non-HTTP cookies are only accessible via "non-HTTP" APIs. For example, a developer might use HTML's "document.cookie" to set a non-HTTP cookie as follows:

```
document.cookie = "name=value; Secure; NonHttp";
```

That cookie would be available via subsequent calls to "document.cookie", but will not be included as a "Cookie" header in HTTP requests to the developer's server.

The same cookie would not be accepted if set via a "Set-Cookie" header. That is, the following header would result in no change to the user agent's cookie store:

```
Set-Cookie: name=value; Secure; NonHttp
```

Likewise, cookies that contain both the "NonHttp" and "HttpOnly" flags will be ignored. The following is a no-op:

```
document.cookie = "name=value; Secure; NonHttp; HttpOnly"
```

As is the following:

```
Set-Cookie: name=value; Secure; NonHttp; HttpOnly
```

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].

3. Recommendations

This section describes extensions to [RFC6265] necessary to implement the "NonHttp" attribute.

1. Extend the "cookie-av" grammar in Section 4.1.1 of [RFC6265] as follows:

```
cookie-av      = expires-av / max-age-av / domain-av /
                 path-av / secure-av / httponly-av /
                 nohttp-av / extension-av
nohttp-av      = "NoHttp"
```

2. Add a section to Section 4.1.2 of [RFC6265] describing the semantics of the "NonHttp" attribute as follows:

   The "NonHttp" attribute limits the scope of the cookie to "non-HTTP" APIs. That is, the attribute instructs the user agent to omit the cookie when constructing a "Cookie" header for an HTTP request.

   If both the "NonHttp" and "HttpOnly" attributes are present when setting a cookie, the cookie will be ignored, regardless of its delivery mechanism.

3. Alter the storage model defined in Section 5.3 of [RFC6265] as follows:

   1. Add "no-http-flag" as a field to be stored on each cookie.

   2. Insert the following two steps after the current step 10:

      1. If the "cookie-attribute-list" contains an attribute with an "attribute-name" of "NoHttp", set the cookie's "no-http-flag" to "true". Otherwise, set the cookie's "no-http-flag" to "false".

      2. If the cookie was not received from a "non-HTTP" API, and the cookie's "no-http-flag" is "true", abort these steps and ignore the cookie entirely.

   3. Insert the following step after the current step 11.2:

      1. If the newly created cookie was not received from a "non-HTTP" API, and the "old-cookie"'s "no-http-flag" is "true", abort these steps and ignore the newly created cookie entirely.

4. Add a section to Section 5.2 of [RFC6265] describing the processing requirements for the "NonHttp" attribute as follows:

If the "attribute-name" case-insensitively matches the string "NonHttp", the user agent MUST append an attribute to the "cookie-attribute-list" with an "attribute-name" of "NonHttp" and an empty "attribute-value".

5. Alter the "Cookie" header generation in Section 5.4 of [RFC6265] as follows:

    1. Add the following requirement to the list of requirements in the current step 1:

        + If the cookie's "no-http-flag" is "true", then exclude the cookie if the "cookie-string" is not being generated for a "non-HTTP" API (as defined by the user agent).

4. Security and Privacy Considerations

4.1. Cross-origin storage capabilities

There's a risk that allowing developers to suppress cookies from HTTP requests might lead to increased usage of cookies as a cross-origin storage mechanism. Given existing usage, however, the worst case seems to be an increase from a high number to a marginally higher number.

Note, though, that the capabilities provided here can already be obtained through clever use of the "path" attribute to scope cookies down to paths that aren't frequently requested. These cookies can be accessed through clever use of "<iframe>" elements and the "history.pushState" API.

The flag introduced here makes that mechanism significantly simpler, but does not introduce fundamentally new capabilities.

4.2. Fewer Cookies on the Wire

Non-HTTP cookies are never sent directly by the user agent over the network, which reduces their exposure to both active and passive network attackers. It seems reasonable to expect that adoption of the "NonHttp" attribute by popular analytics packages could result in a substantial reduction in the usefulness of those packages' cookies when attempting to correlate a given user's activities over time and across networks.

4.3. Legacy Clients

Clients that do not support the "NonHttp" flag will fall back to the existing behavior: cookies with the flag will be accepted via any delivery mechanism, and will continue to be sent out with matching HTTP requests.

Developers can use this behavior to feature-detect support for the flag: sending "Set-Cookie: nonhttp-support=0; NonHttp" will allow server-side detection by parsing the "Cookie" header sent with requests, and client-side detection via code like "document.cookie.match('nonhttp-support=0')".

4.4. Other Constraints

Whether or not a cookie is flagged as "NonHttp" MUST not affect eviction behavior in general. Cookies are cookies, no matter their flags, so the "remove excess cookies" constraints, expiration behavior, and "persistent-flag" behavior discussed in the current step 12 of Section 5.3 of [RFC6265] would still apply.

5. Aesthetic Considerations

5.1. Isn't 'DOMOnly' a better name? Or 'NoHttp'? Or 'Insert Bikeshed Here'?

"DOMOnly" is probably a better fit for the practical implications of this API. "DocumentCookieOnly" is even more clear. This document runs with "NonHttp" to match RFC 6265's existing language around "non-HTTP" APIs, and to avoid arguments about software that handles cookies, but doesn't work with a DOM or any particular JavaScript interface. Bikeshedding is, of course, welcome.

6. Normative References

[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>.

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

Appendix A. Acknowledgements

Michael Nordman suggested this approach during a conversation about cross-origin storage mechanisms. Brad Townsend helped me understand the potential positive impact on traffic levels for analytics customers.

Author's Address

Mike West
Google, Inc

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