Internet Engineering Task Force (IETF)
Request for Comments: 6454
Category: Standards Track
ISSN: 2070-1721

A. Barth
Google, Inc.

December 2011

The Web Origin Concept

Abstract

This document defines the concept of an "origin", which is often used as the scope of authority or privilege by user agents. Typically, user agents isolate content retrieved from different origins to prevent malicious web site operators from interfering with the operation of benign web sites. In addition to outlining the principles that underlie the concept of origin, this document details how to determine the origin of a URI and how to serialize an origin into a string. It also defines an HTTP header field, named "Origin", that indicates which origins are associated with an HTTP request.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6454.

Table of Contents

1. Introduction

User agents interact with content created by a large number of authors. Although many of those authors are well-meaning, some authors might be malicious. To the extent that user agents undertake actions based on content they process, user agent implementors might wish to restrict the ability of malicious authors to disrupt the confidentiality or integrity of other content or servers.

As an example, consider an HTTP user agent that renders HTML content retrieved from various servers. If the user agent executes scripts contained in those documents, the user agent implementor might wish to prevent scripts retrieved from a malicious server from reading documents stored on an honest server, which might, for example, be behind a firewall.

Traditionally, user agents have divided content according to its "origin". More specifically, user agents allow content retrieved from one origin to interact freely with other content retrieved from that origin, but user agents restrict how that content can interact with content from another origin.

This document describes the principles behind the so-called same-origin policy as well as the "nuts and bolts" of comparing and serializing origins. This document does not describe all the facets of the same-origin policy, the details of which are left to other specifications, such as HTML [HTML] and WebSockets [RFC6455], because the details are often application-specific.

2. Conventions

2.1. Conformance Criteria

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].

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), CTL (controls), DIGIT (decimal 0-9), DQUOTE (double quote), HEXDIG (hexadecimal 0-9/A-F/a-f), LF (line feed), OCTET (any 8-bit sequence of data), SP (space), HTAB (horizontal tab), CHAR (any US-ASCII character), VCHAR (any visible US-ASCII character), and WSP (whitespace).

The OWS rule is used where zero or more linear whitespace octets might appear. OWS SHOULD either not be produced or be produced as a single SP. Multiple OWS octets that occur within field-content SHOULD either be replaced with a single SP or transformed to all SP octets (each octet other than SP replaced with SP) before interpreting the field value or forwarding the message downstream.

```
OWS            = *( SP / HTAB / obs-fold )
                 ; "optional" whitespace
obs-fold       = CRLF ( SP / HTAB )
                 ; obsolete line folding
```

2.3. Terminology

The terms "user agent", "client", "server", "proxy", and "origin server" have the same meaning as in the HTTP/1.1 specification ([RFC2616], Section 1.3).

A globally unique identifier is a value that is different from all other previously existing values. For example, a sufficiently long random string is likely to be a globally unique identifier. If the origin value never leaves the user agent, a monotonically increasing counter local to the user agent can also serve as a globally unique identifier.

3. Principles of the Same-Origin Policy

Many user agents undertake actions on behalf of remote parties. For example, HTTP user agents follow redirects, which are instructions from remote servers, and HTML user agents expose rich Document Object Model (DOM) interfaces to scripts retrieved from remote servers.

Without any security model, user agents might undertake actions detrimental to the user or to other parties. Over time, many web-related technologies have converged towards a common security model,

known colloquially as the "same-origin policy". Although this security model evolved largely organically, the same-origin policy can be understood in terms of a handful of key concepts. This section presents those concepts and provides advice about how to use these concepts securely.

3.1. Trust

The same-origin policy specifies trust by URI. For example, HTML documents designate which script to run with a URI:

```
<script src="https://example.com/library.js"></script>
```

When a user agent processes this element, the user agent will fetch the script at the designated URI and execute the script with the privileges of the document. In this way, the document grants all the privileges it has to the resource designated by the URI. In essence, the document declares that it trusts the integrity of information retrieved from that URI.

In addition to importing libraries from URIs, user agents also send information to remote parties designated by URI. For example, consider the HTML form element:

```
<form method="POST" action="https://example.com/login">
 ... <input type="password"> ...
</form>
```

When the user enters his or her password and submits the form, the user agent sends the password to the network endpoint designated by the URI. In this way, the document exports its secret data to that URI, in essence declaring that it trusts the confidentiality of information sent to that URI.

3.1.1. Pitfalls

When designing new protocols that use the same-origin policy, make sure that important trust distinctions are visible in URIs. For example, if both Transport Layer Security (TLS) and non-TLS protected resources use the "http" URI scheme (as in [RFC2817]), a document would be unable to specify that it wishes to retrieve a script only over TLS. By using the "https" URI scheme, documents are able to indicate that they wish to interact with resources that are protected from active network attackers.

3.2. Origin

In principle, user agents could treat every URI as a separate protection domain and require explicit consent for content retrieved from one URI to interact with another URI. Unfortunately, this design is cumbersome for developers because web applications often consist of a number of resources acting in concert.

Instead, user agents group URIs together into protection domains called "origins". Roughly speaking, two URIs are part of the same origin (i.e., represent the same principal) if they have the same scheme, host, and port. (See Section 4 for full details.)

Q: Why not just use the host?

A: Including the scheme in the origin tuple is essential for security. If user agents did not include the scheme, there would be no isolation between http://example.com and https://example.com because the two have the same host. However, without this isolation, an active network attacker could corrupt content retrieved from http://example.com and have that content instruct the user agent to compromise the confidentiality and integrity of content retrieved from https://example.com, bypassing the protections afforded by TLS [RFC5246].

Q: Why use the fully qualified host name instead of just the "top-level" domain?

A: Although the DNS has hierarchical delegation, the trust relationships between host names vary by deployment. For example, at many educational institutions, students can host content at https://example.edu/~student/, but that does not mean a document authored by a student should be part of the same origin (i.e., inhabit the same protection domain) as a web application for managing grades hosted at https://grades.example.edu/.

The example.edu deployment illustrates that grouping resources by origin does not always align perfectly with every deployment scenario. In this deployment, every student's web site inhabits the same origin, which might not be desirable. In some sense, the origin granularity is a historical artifact of how the security model evolved.

3.2.1. Examples

All of the following resources have the same origin:

http://example.com/
http://example.com:80/
http://example.com/path/file

Each of the URIs has the same scheme, host, and port components.

Each of the following resources has a different origin from the others.

http://example.com/
http://example.com:8080/
http://www.example.com/
https://example.com:80/
https://example.com/
http://example.org/
http://ietf.org/

In each case, at least one of the scheme, host, and port component will differ from the others in the list.

3.3. Authority

Although user agents group URIs into origins, not every resource in an origin carries the same authority (in the security sense of the word "authority", not in the [RFC3986] sense). For example, an image is passive content and, therefore, carries no authority, meaning the image has no access to the objects and resources available to its origin. By contrast, an HTML document carries the full authority of its origin, and scripts within (or imported into) the document can access every resource in its origin.

User agents determine how much authority to grant a resource by examining its media type. For example, resources with a media type of image/png are treated as images, and resources with a media type of text/html are treated as HTML documents.

When hosting untrusted content (such as user-generated content), web applications can limit that content’s authority by restricting its media type. For example, serving user-generated content as image/png is less risky than serving user-generated content as text/html. Of course, many web applications incorporate untrusted content in their HTML documents. If not done carefully, these applications risk leaking their origin’s authority to the untrusted content, a vulnerability commonly known as cross-site scripting.

3.3.1. Pitfalls

When designing new pieces of the web platform, be careful not to grant authority to resources irrespective of media type. Many web applications serve untrusted content with restricted media types. A new web platform feature that grants authority to these pieces of content risks introducing vulnerabilities into existing applications. Instead, prefer to grant authority to media types that already possess the origin's full authority or to new media types designed specifically to carry the new authority.

In order to remain compatible with servers that supply incorrect media types, some user agents employ "content sniffing" and treat content as if it had a different media type than the media type supplied by the server. If not done carefully, content sniffing can lead to security vulnerabilities because user agents might grant low-authority media types, such as images, the privileges of high-authority media types, such as HTML documents [SNIFF].

3.4. Policy

Generally speaking, user agents isolate different origins and permit controlled communication between origins. The details of how user agents provide isolation and communication vary depending on several factors.

3.4.1. Object Access

Most objects (also known as application programming interfaces or APIs) exposed by the user agent are available only to the same origin. Specifically, content retrieved from one URI can access objects associated with content retrieved from another URI if, and only if, the two URIs belong to the same origin, e.g., have the same scheme, host, and port.

There are some exceptions to this general rule. For example, some parts of HTML's Location interface are available across origins (e.g., to allow for navigating other browsing contexts). As another example, HTML's postMessage interface is visible across origins explicitly to facilitate cross-origin communication. Exposing objects to foreign origins is dangerous and should be done only with great care because doing so exposes these objects to potential attackers.

3.4.2. Network Access

Access to network resources varies depending on whether the resources are in the same origin as the content attempting to access them.

Generally, reading information from another origin is forbidden. However, an origin is permitted to use some kinds of resources retrieved from other origins. For example, an origin is permitted to execute script, render images, and apply style sheets from any origin. Likewise, an origin can display content from another origin, such as an HTML document in an HTML frame. Network resources can also opt into letting other origins read their information, for example, using Cross-Origin Resource Sharing [CORS]. In these cases, access is typically granted on a per-origin basis.

Sending information to another origin is permitted. However, sending information over the network in arbitrary formats is dangerous. For this reason, user agents restrict documents to sending information using particular protocols, such as in an HTTP request without custom headers. Expanding the set of allowed protocols, for example, by adding support for WebSockets, must be done carefully to avoid introducing vulnerabilities [RFC6455].

3.4.3. Pitfalls

Whenever user agents allow one origin to interact with resources from another origin, they invite security issues. For example, the ability to display images from another origin leaks their height and width. Similarly, the ability to send network requests to another origin gives rise to cross-site request forgery vulnerabilities [CSRF]. However, user agent implementors often balance these risks against the benefits of allowing the cross-origin interaction. For example, an HTML user agent that blocked cross-origin network requests would prevent its users from following hyperlinks, a core feature of the web.

When adding new functionality to the web platform, it can be tempting to grant a privilege to one resource but to withhold that privilege from another resource in the same origin. However, withholding privileges in this way is ineffective because the resource without the privilege can usually obtain the privilege anyway because user agents do not isolate resources within an origin. Instead, privileges should be granted or withheld from origins as a whole (rather than discriminating between individual resources within an origin) [BOFGO].

3.5. Conclusion

The same-origin policy uses URIs to designate trust relationships. URIs are grouped together into origins, which represent protection domains. Some resources in an origin (e.g., active content) are granted the origin's full authority, whereas other resources in the origin (e.g., passive content) are not granted the origin's authority. Content that carries its origin's authority is granted access to objects and network resources within its own origin. This content is also granted limited access to objects and network resources of other origins, but these cross-origin privileges must be designed carefully to avoid security vulnerabilities.

4. Origin of a URI

The origin of a URI is the value computed by the following algorithm:

1. If the URI does not use a hierarchical element as a naming authority (see [RFC3986], Section 3.2) or if the URI is not an absolute URI, then generate a fresh globally unique identifier and return that value.

   NOTE: Running this algorithm multiple times for the same URI can produce different values each time. Typically, user agents compute the origin of, for example, an HTML document once and use that origin for subsequent security checks rather than recomputing the origin for each security check.

2. Let uri-scheme be the scheme component of the URI, converted to lowercase.

3. If the implementation doesn't support the protocol given by uri-scheme, then generate a fresh globally unique identifier and return that value.

4. If uri-scheme is "file", the implementation MAY return an implementation-defined value.

   NOTE: Historically, user agents have granted content from the file scheme a tremendous amount of privilege. However, granting all local files such wide privileges can lead to privilege escalation attacks. Some user agents have had success granting local files directory-based privileges, but this approach has not been widely adopted. Other user agents use globally unique identifiers for each file URI, which is the most secure option.

5. Let uri-host be the host component of the URI, converted to lower case (using the i;ascii-casemap collation defined in [RFC4790]).

   NOTE: This document assumes that the user agent performs Internationalizing Domain Names in Applications (IDNA) processing and validation when constructing the URI. In particular, this document assumes the uri-host will contain only LDH labels because the user agent will have already converted any non-ASCII labels to their corresponding A-labels (see [RFC5890]). For this reason, origin-based security policies are sensitive to the IDNA algorithm employed by the user agent. See Section 8.4 for further discussion.

6. If there is no port component of the URI:

   1. Let uri-port be the default port for the protocol given by uri-scheme.

   Otherwise:

   2. Let uri-port be the port component of the URI.

7. Return the triple (uri-scheme, uri-host, uri-port).

5. Comparing Origins

Two origins are "the same" if, and only if, they are identical. In particular:

- If the two origins are scheme/host/port triples, the two origins are the same if, and only if, they have identical schemes, hosts, and ports.

- An origin that is a globally unique identifier cannot be the same as an origin that is a scheme/host/port triple.

Two URIs are same-origin if their origins are the same.

   NOTE: A URI is not necessarily same-origin with itself. For example, a data URI [RFC2397] is not same-origin with itself because data URIs do not use a server-based naming authority and therefore have globally unique identifiers as origins.

6. Serializing Origins

This section defines how to serialize an origin to a unicode [Unicode6] string and to an ASCII [RFC20] string.

6.1. Unicode Serialization of an Origin

The unicode-serialization of an origin is the value returned by the following algorithm:

1. If the origin is not a scheme/host/port triple, then return the string

   null

   (i.e., the code point sequence U+006E, U+0075, U+006C, U+006C) and abort these steps.

2. Otherwise, let result be the scheme part of the origin triple.

3. Append the string "://" to result.

4. Append each component of the host part of the origin triple (converted as follows) to the result, separated by U+002E FULL STOP code points (".").

   1. If the component is an A-label, use the corresponding U-label instead (see [RFC5890] and [RFC5891]).

   2. Otherwise, use the component verbatim.

5. If the port part of the origin triple is different from the default port for the protocol given by the scheme part of the origin triple:

   1. Append a U+003A COLON code point (":") and the given port, in base ten, to result.

6. Return result.

6.2. ASCII Serialization of an Origin

The ascii-serialization of an origin is the value returned by the following algorithm:

1. If the origin is not a scheme/host/port triple, then return the string

   null

   (i.e., the code point sequence U+006E, U+0075, U+006C, U+006C) and abort these steps.

2. Otherwise, let result be the scheme part of the origin triple.

3. Append the string "://" to result.

4. Append the host part of the origin triple to result.

5. If the port part of the origin triple is different from the default port for the protocol given by the scheme part of the origin triple:

    1. Append a U+003A COLON code point (":") and the given port, in base ten, to result.

6. Return result.

7. The HTTP Origin Header Field

This section defines the HTTP Origin header field.

7.1. Syntax

The Origin header field has the following syntax:

```
origin              = "Origin:" OWS origin-list-or-null OWS
origin-list-or-null = %x6E %x75 %x6C %x6C / origin-list
origin-list         = serialized-origin *( SP serialized-origin )
serialized-origin   = scheme "://" host [ ":" port ]
                    ; <scheme>, <host>, <port> from RFC 3986
```

7.2. Semantics

When included in an HTTP request, the Origin header field indicates the origin(s) that "caused" the user agent to issue the request, as defined by the API that triggered the user agent to issue the request.

For example, consider a user agent that executes scripts on behalf of origins. If one of those scripts causes the user agent to issue an HTTP request, the user agent MAY use the Origin header field to inform the server of the security context in which the script was executing when it caused the user agent to issue the request.

In some cases, a number of origins contribute to causing the user agents to issue an HTTP request. In those cases, the user agent MAY list all the origins in the Origin header field. For example, if the HTTP request was initially issued by one origin but then later

redirected by another origin, the user agent MAY inform the server that two origins were involved in causing the user agent to issue the request.

7.3. User Agent Requirements

The user agent MAY include an Origin header field in any HTTP request.

The user agent MUST NOT include more than one Origin header field in any HTTP request.

Whenever a user agent issues an HTTP request from a "privacy-sensitive" context, the user agent MUST send the value "null" in the Origin header field.

> NOTE: This document does not define the notion of a privacy-sensitive context. Applications that generate HTTP requests can designate contexts as privacy-sensitive to impose restrictions on how user agents generate Origin header fields.

When generating an Origin header field, the user agent MUST meet the following requirements:

- Each of the serialized-origin productions in the grammar MUST be the ascii-serialization of an origin.
- No two consecutive serialized-origin productions in the grammar can be identical. In particular, if the user agent would generate two consecutive serialized-origins, the user agent MUST NOT generate the second one.

8. Security Considerations

The same-origin policy is one of the cornerstones of security for many user agents, including web browsers. Historically, some user agents tried other security models, including taint tracking and exfiltration prevention, but those models proved difficult to implement at the time (although there has been recent interest in reviving some of these ideas).

Evaluating the security of the same-origin policy is difficult because the origin concept itself plays such a central role in the security landscape. The notional origin itself is just a unit of isolation, imperfect as are most one-size-fits-all notions. That said, there are some systemic weaknesses, discussed below.

8.1. Reliance on DNS

In practice, the same-origin policy relies upon the Domain Name System (DNS) for security because many commonly used URI schemes, such as http, use DNS-based naming authorities. If the DNS is partially or fully compromised, the same-origin policy might fail to provide the security properties required by applications.

Some URI schemes, such as https, are more resistant to DNS compromise because user agents employ other mechanisms, such as certificates, to verify the source of content retrieved from these URIs. Other URI schemes, such as the chrome-extension URI scheme (see Section 4.3 of [CRX]), use a public-key-based naming authority and are fully secure against DNS compromise.

The web origin concept isolates content retrieved from different URI schemes; this is essential to containing the effects of DNS compromise.

8.2. Divergent Units of Isolation

Over time, a number of technologies have converged on the web origin concept as a convenient unit of isolation. However, many technologies in use today, such as cookies [RFC6265], pre-date the modern web origin concept. These technologies often have different isolation units, leading to vulnerabilities.

One alternative is to use only the "registry-controlled" domain rather than the fully qualified domain name as the unit of isolation (e.g., "example.com" instead of "www.example.com"). This practice is problematic for a number of reasons and is NOT RECOMMENDED:

1. The notion of a "registry-controlled" domain is a function of human practice surrounding the DNS rather than a property of the DNS itself. For example, many municipalities in Japan run public registries quite deep in the DNS hierarchy. There are widely used "public suffix lists", but these lists are difficult to keep up to date and vary between implementations.

2. This practice is incompatible with URI schemes that do not use a DNS-based naming authority. For example, if a given URI scheme uses public keys as naming authorities, the notion of a "registry-controlled" public key is somewhat incoherent. Worse, some URI schemes, such as nntp, use dotted delegation in the opposite direction from DNS (e.g., alt.usenet.kooks), and others use the DNS but present the labels in the reverse of the usual order (e.g., com.example.www).

At best, using "registry-controlled" domains is URI-scheme- and implementation-specific. At worst, differences between URI schemes and implementations can lead to vulnerabilities.

8.3. Ambient Authority

When using the same-origin policy, user agents grant authority to content based on its URI rather than based on which objects the content can designate. This disentangling of designation from authority is an example of ambient authority and can lead to vulnerabilities.

Consider, for example, cross-site scripting in HTML documents. If an attacker can inject script content into an HTML document, those scripts will run with the authority of the document's origin, perhaps allowing the script access to sensitive information, such as the user's medical records. If, however, the script's authority were limited to those objects that the script could designate, the attacker would not gain any advantage by injecting the script into an HTML document hosted by a third party.

8.4. IDNA Dependency and Migration

The security properties of the same-origin policy can depend crucially on details of the IDNA algorithm employed by the user agent. In particular, a user agent might map some international domain names (for example, those involving the U+00DF character) to different ASCII representations depending on whether the user agent uses IDNA2003 [RFC3490] or IDNA2008 [RFC5890].

Migrating from one IDNA algorithm to another might redraw a number of security boundaries, potentially erecting new security boundaries or, worse, tearing down security boundaries between two mutually distrusting entities. Changing security boundaries is risky because combining two mutually distrusting entities into the same origin might allow one to attack the other.

9. IANA Considerations

The permanent message header field registry (see [RFC3864]) has been updated with the following registration:

Header field name: Origin

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document: this specification (Section 7)

10. References

10.1. Normative References

[RFC20] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4790] Newman, C., Duerst, M., and A. Gulbrandsen, "Internet Application Protocol Collation Registry", RFC 4790, March 2007.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.

[Unicode6] The Unicode Consortium, "The Unicode Standard, Version 6.0.0", 2011, <http://www.unicode.org/versions/Unicode6.0.0/>.

10.2. Informative References

[BOFGO] Jackson, C. and A. Barth, "Beware of Finer-Grained Origins", 2008, <http://w2spconf.com/2008/papers/s2p1.pdf>.

[CORS] van Kesteren, A., "Cross-Origin Resource Sharing", W3C Working Draft WD-cors-20100727, July 2010, <http://www.w3.org/TR/2010/WD-cors-20100727/>.

Latest version available at <http://www.w3.org/TR/cors/>.

[CRX] Barth, A., Felt, A., Saxena, P., and A. Boodman, "Protecting Browsers from Extension Vulnerabilities", 2010, <http://www.isoc.org/isoc/conferences/ndss/10/pdf/04.pdf>.

[CSRF] Barth, A., Jackson, C., and J. Mitchell, "Robust Defenses for Cross-Site Request Forgery", 2008, <http://portal.acm.org/citation.cfm?id=1455770.1455782>.

[HTML] Hickson, I., "HTML5", W3C Working Draft WD-html5-20110525, May 2011, <http://www.w3.org/TR/2011/WD-html5-20110525/>.

Latest version available at <http://www.w3.org/TR/html5/>.

[RFC2397] Masinter, L., "The "data" URL scheme", RFC 2397, August 1998.

[RFC2817] Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, May 2000.

[RFC3490] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

[SNIFF] Barth, A. and I. Hickson, "Media Type Sniffing", Work in Progress, May 2011.

Appendix A. Acknowledgements

We would like to thank Lucas Adamski, Stephen Farrell, Miguel A. Garcia, Tobias Gondrom, Ian Hickson, Anne van Kesteren, Jeff Hodges, Collin Jackson, Larry Masinter, Alexey Melnikov, Mark Nottingham, Julian Reschke, Peter Saint-Andre, Jonas Sicking, Sid Stamm, Daniel Veditz, and Chris Weber for their valuable feedback on this document.

Author's Address

Adam Barth
Google, Inc.

EMail: ietf@adambarth.com
URI: http://www.adambarth.com/