Working Group	A. Barth
Internet-Draft	U.C. Berkeley
Expires: December 11, 2010	C. Jackson
	Stanford University
	I. Hickson
	Google, Inc.
	June 9, 2010

TOC

The Web Origin Concept
draft-abarth-origin-07

Abstract

This document defines the concept of an "origin," which is used by web browsers to isolate content retrieved from different parties. The origin concept is defined by a "same-origin" relation and a serialization algorithm. This document also defines an HTTP Origin header, which a user agent can use to describe the security contexts that caused the user agent to initiate an HTTP request. HTTP servers can use the Origin header to mitigate against Cross-Site Request Forgery (CSRF) vulnerabilities.

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 11, 2010.

Table of Contents

---

1. Introduction

TOC

This document defines the concept of an "origin," which is used by web browsers to isolate content retrieved from different parties. The origin concept is defined by a "same-origin" relation and a serialization algorithm. This document also defines an HTTP Origin header, which a user agent can use to describe the security contexts that caused the user agent to initiate an HTTP request. HTTP servers can use the Origin header to mitigate against Cross-Site Request Forgery (CSRF) vulnerabilities.
TODO: Discuss other CSRF defenses.

---

2. Origin

TOC

The following algorithm MUST be used to compute the origin of a URI.

1. Let /uri/ be the URI for which the origin is being determined.

2. Parse /uri/.

3. If /uri/ does not use a server-based naming authority, or if parsing /uri/ failed, or if /uri/ is not an absolute URI, then return an implementation-defined value.

4. Let /scheme/ be the scheme component of /uri/, converted to lowercase.

5. If the implementation doesn't support the protocol given by /scheme/, then return an implementation-defined value.

6. If /scheme/ is "file", then the implementation MAY return a implementation-defined value.

7. Let /host/ be the host component of /uri/.

8. Apply the IDNA ToASCII algorithm [RFC3490] to /host/, with both the AllowUnassigned and UseSTD3ASCIIRules flags set. Let /host/ be the result of the ToASCII algorithm.

9. If ToASCII fails to convert one of the components of the string (e.g. because it is too long or because it contains invalid characters), then return an implementation-defined value.

10. Let /host/ be the result of converting /host/ to lowercase.

11. If there is no port component of /uri/, then let /port/ be the default port for the protocol given by /scheme/. Otherwise, let /port/ be the port component of /uri/.

12. Return the tuple (/scheme/, /host/, /port/).

Implementations MAY define other types of origins in addition to the scheme/host/port tuple type defined above. (For example, user agents could implement globally unique origins or certificate-based origins.)

---

3. Comparing Origins

TOC

Implementations MUST use the following algorithm to test whether two origins are the "same origin".

1. Let /A/ be the first origin being compared, and let /B/ be the second origin being compared.

2. If either /A/ or /B/ is not a scheme/host/port tuple, return an implementation-defined value.

3. If /A/ and /B/ have scheme components that are not identical, return false.

4. If /A/ and /B/ have host components that are not identical, return false.

5. If /A/ and /B/ have port components that are not identical, return false.

6. Return true.

---

4. Serializing Origins

TOC

4.1. Unicode Serialization of an Origin

TOC

Implementations MUST using the following algorithm to compute the Unicode serialization of an origin:

1. If the origin in question is not a scheme/host/port tuple, 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 tuple.

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

4. Apply the IDNA ToUnicode algorithm [RFC3490] to each component of the host part of the origin tuple, and append the results of each component, in the same order, separated by U+002E FULL STOP code points (".") to /result/.

5. If the port part of the origin tuple gives a port that is different from the default port for the protocol given by the scheme part of the origin tuple, then append a U+003A COLON code points (":") and the given port, in base ten, to /result/.

6. Return /result/.

TODO: Check that we handle IPv6 literals correctly.

4.2. ASCII Serialization of an Origin

TOC

Implementations MUST using the following algorithm to compute the ASCII serialization of an origin:

1. If the origin in question is not a scheme/host/port tuple, 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 tuple.

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

4. If the host part of the origin tuple is in the form of a DNS domain name, apply the IDNA conversion algorithm ([RFC3490] section 4) to it, with both the AllowUnassigned and UseSTD3ASCIIRules flags set, employ the ToASCII operation, and append the result to /result/.

5. If ToASCII fails to convert one of the components of the string, e.g. because it is too long or because it contains invalid characters, then return the literal string "null" and abort these steps.

6. If the port part of the origin tuple gives a port that is different from the default port for the protocol given by the scheme part of the origin tuple, then append a U+003A COLON code point (":") and the given port, in base ten, to /result/.

7. Return /result/.

5. User Agent Behavior

TOC

Whenever a user agent issues an HTTP request, the user agent MUST include exactly one HTTP header named "Origin" that conforms to the following ABNF [RFC5234] grammar:

```
origin              = "origin" ":" origin-list-or-null
origin-list-or-null = OWS [ "null" / origin-list ] OWS
origin-list         = serialized-origin *( SP serialized-origin )
serialized-origin   = scheme "://" host [ ":" port ]
 ; <scheme>, <host>, <port> productions from RFC3986
```

Whenever a user agent would send a Origin header containing two consecutive, identical origin serializations, the user agent MUST remove one such origin serialization from the header.
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.
If /B/ is the Request-URI in the Request-Line of an HTTP request, then the associated HTTP response is an "HTTP redirect from URI /B/" if the response contains a 3xx Status Code (all terms RFC2616).
Whenever a user agent issues an HTTP request to URI /A/ as a result of an HTTP redirect from URI /B/, the user agent MUST either:

1. set the value of the Origin header in the HTTP request to /A/ to "null" (i.e., the code point sequence U+006E, U+0075, U+006C, U+006C),

2. set the value of the Origin header in the /A/ request to the value of the Origin header in the /B/ request extended with a space and the ASCII serialization of the origin of /B/, unless this would result in the header containing the origin serialization "null" in a component.

Whenever a user agent issues an HTTP request that (1) is *not* the result of an HTTP redirect and (2) is *not* initiated from a "privacy-sensitive" context, the user agent SHOULD set the value of the Origin header to the ASCII serialization of the origin that initiated the HTTP request.
Note: This behavior differs from that of the HTTP Referer header, which user agents often suppress when an origin with an "https" scheme issues a request for a URI with an "http" scheme.

---

6. HTTP Server Behavior

TOC

HTTP Servers MAY use the Origin header to "defend themselves against CSRF attacks." Such servers are known as "participating servers" in this section.
Let the /origin white list/ of a participating server be a set of strings selected by the operator of that server.
The string "null" MUST NOT be a member of the /origin white list/ for any participating server.

> Example: The origin white list for the example.com Web server could be the strings "http://example.com", "https://example.com", "http://www.example.com", and "https://www.example.com".

A participating server MUST use the following algorithm when determining whether to modify state in response to an HTTP request:

1. If the request method is safe (as defined by RFC 2616, Section 9.1.1, e.g. either "GET" no "HEAD"), return "MUST NOT modify state" and abort these steps.

2. If the request does not contain a header named "Origin", return "MAY modify state" abort these steps.

3. For each request header named "Origin", let the /initiating origin list/ be the list of origins represented in the header:

    1. If there exists a origin in the /initiating origin list/ is not a member of the /origin white list/ for this server, return "MUST NOT modify state" and abort these steps.

4. Return "MAY modify state".

> Example: A Web server could modify state in response to POST requests that lack an Origin header (because these requests are sent by non-supporting user agents) and could modify state in response to POST requests that have an Origin header of "http://example.com", "https://example.com", "http://www.example.com", or "https://www.example.com".

---

7. Privacy Considerations

TOC

This section is not normative.
The Origin header improves on the Referer header by respecting the user's privacy: The Origin header includes only the information required to identify the principal that initiated the request (typically the scheme, host, and port of initiating origin). In particular, the Origin header does not contain the path or query portions of the URI included in the Referer header that invade privacy without providing additional security.
The Origin header also improves on the Referer header by not leaking intranet host names to external web sites when a user follows a hyperlink from an intranet host to an external site because hyperlinks generate privacy-sensitive requests.

8. Security Considerations

TOC

This section is not normative.
Because a supporting user agent will always include the Origin header when making HTTP requests, HTTP servers can detect that a request was initiated by a supporting user agent by observing the presence of the header. This design prevents a malicious web site from making a supporting user agent appear to be a non-supporting user agent. Unlike the Referer header, which is absent when suppressed by the user agent, the Origin header takes on the value "null" when suppressed by the user agent.
In some legacy user agents, The Origin header can be spoofed for same-site XMLHttpRequests. Sites that rely only on network connectivity for authentication should use a DNS rebinding defense, such as validating the HTTP Host header, in addition to CSRF protection.

9. IANA Considerations

TOC

TODO: The "Origin" header should be registered.

10. TODO

TOC

Think about how this interacts with proxies.
Think about how this interacts with caches.
Think about how this interacts with IPv6.

Authors' Addresses

TOC

	Adam Barth
	University of California, Berkeley
Email:	abarth@eecs.berkeley.edu
URI:	http://www.adambarth.com/
	Collin Jackson
	Stanford University
Email:	collinj@cs.stanford.edu
URI:	http://www.collinjackson.com/
	Ian Hickson
	Google, Inc.

Email:	ian@hixie.ch
URI:	http://ln.hixie.ch/