Working Group	A. Barth	
Internet-Draft	U.C. Berkeley	
Expires: July 26, 2009	C. Jackson	
	Stanford University	
	I. Hickson	
	Google, Inc.	
	January 22, 2009	

TOC

The HTTP Origin Header
draft-abarth-origin-00

Status of this Memo

Copyright Notice

Abstract

This document defines the HTTP Origin header. The Origin header is added by the user agent to describe the security context that initiated an HTTP request. HTTP servers can use the Origin header to defend themselves against Cross-Site Request Forgery (CSRF) attacks.

Table of Contents

1. Introduction

TOC

This document describes the HTTP Origin header. The Origin header identifies the security context that initiated an HTTP request and can be used by Web sites to mitigate cross-site request forgery (CSRF) vulnerabilities.

2. Origin

TOC

The following algoritm MUST be used to compute the origin of a URL.

1. Let /url/ be the URL for which the origin is being determined.

2. Parse /url/.

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

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

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

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

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

8. Apply the IDNA ToASCII algorithm 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 /url/, then let /port/ be the default port for the protocol given by /scheme/. Otherwise, let /port/ be the port component of /url/.

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 character 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 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 characters (".") 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 character (":") and the given port, in base ten, to /result/.

6. Return /result/.

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 character 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 ToASCII algorithm the host part of the origin tuple, with both the AllowUnassigned and UseSTD3ASCIIRules flags set, 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 character (":") 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 MAY include an HTTP header named "Origin".
Whenever a user agent issues an HTTP request whose method is neither "GET" nor "HEAD", the user agent MUST include exactly one HTTP header named "Origin".
Whenever a user agent issues an HTTP request that contains an HTTP header named "Origin", the value of that header MUST either be

1. the string "null" (i.e., the character sequence U+006E, U+0075, U+006C, U+006C) or

2. the ASCII serialization of the origin that initiated the HTTP request.

Whenever a user agent issues an HTTP request that contains an HTTP header named "Origin", if the request was initiated on behalf of an origin, the user agent SHOULD use the ASCII serialization of that origin as the value of the Origin header.

> 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 URL with an "http" scheme.

If a user agent issues an HTTP request in reaction to an HTTP redirect, the Origin header MUST contain the same value as the Origin header in the HTTP request that generated the redirect.

---

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 "GET", return "MUST NOT modify state" and abort these steps.

2. If the request method is "HEAD", return "MUST NOT modify state" and abort these steps.

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

4. For each request header named "Origin", let /initiating origin/ be the value of the header:

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

5. 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".

A participating server MUST NOT instruct a user agent to issue an HTTP request for a given URL unless the following algorithm returns "Safe".

1. If the request method is "GET", return "Safe" and abort these steps.
2. If the request method is "HEAD", return "Safe" and abort these steps.
3. Let /url/ be the URL in question.
4. Let /target origin/ be the origin of /url/.
5. If the ASCII serialization of /target origin/ is a member of the server's /origin white list/, then return "Safe" and abort these steps.
6. Return "Unsafe".

> Example: A Web server would be vulnerable to a CSRF attack if it responded to an HTTP request with HTML that generated a POST request to http://attacker.com/ because the attacker's server could respond with an HTTP 307 status and redirect the POST back to the original server.

---

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 URL 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 GET requests.

TOC

8. Security Considerations

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 an attacker 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 existing 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.

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/