~~CORS Sec Cons~~

4 Security Considerations

This section is non-normative.

Security requirements and considerations are listed throughout this specification. This section lists advice that did not fit anywhere else.

Historically, cross-origin interactions, through hyperlinks, transclusion and form submission, were the most important and distinguishing features of HTTP, HTML and the World Wide Web. As the Web moved from being composed of static markup and resources rendered by the user agent to also include active content, through plug-ins and embedded scripting, and client-side state, through cookies, it quickly became clear that unrestricted cross-origin interactions presented serious privacy and security risks. To deal with these risks, user agents and plugin technologies introduced a set of restrictions generally known as the Same Origin Policy. (SOP) Though many variants of the SOP exist, they all generally 1) preserved the existing cross-origin interactions allowed by HTML and HTTP while 2) restricting the ability of active content to read or make new types of requests across origins.

This specification allows resources to voluntarily relax these restrictions. To do so safely, it is important to understand both 1) the pre-existing security impacts of cross-origin requests allowed by the legacy architecture of the Web as well as 2) the distinction between the goal of this specification: authorizing cross-origin read access to a resource in the user agent, and the possibly unintended consequences of authorizing write/execute access to resources by applications from foreign origins executing in the user agent.

4.1 Simple Requests

In this specification, ~~A~~ simple ~~cross-origin~~ requests~~t~~ ~~has been~~are defined as ~~congruent~~ the set of HTTP methods, headers and data which may ~~with those which may~~ be ~~generated~~ sent cross-origin by currently deployed user agents that do not implement CORS. ~~not conform to this specification. Simple cross-origin requests generated outside this specification (such as~~ These include ~~cross-origin~~ form submissions using `GET` or `POST`, cross-origin hyperlink dereferencing, resource transclusion (as with the `img` tag), ~~and~~and~~or~~ the special case of cross-origin `GET` requests resulting from the HTML `script` element~~s)~~. Because such cross-origin requests are commonplace, they do not require a preflight request. Simple cross-origin requests generated by user agents through means other than CORS typically always include user credentials, so resources conforming to this specification must always be prepared to expect simple cross-origin requests with credentials.

Because of this, and independently of the existence of CORS, all resources for which simple requests have significance other than retrieval must protect themselves from

Cross-Site Request Forgery (CSRF) by requiring the inclusion of an unguessable token in the explicitly provided content of the request.[CSRF]

4.2 Allowing or Denying All Origins

This specification defines how to authorize an instance of an application from a foreign origin, executing in the user agent, to access the representation of the resource in an HTTP response. However, c~~C~~ertain types of resources should not attempt to specify particular authorized origins, but instead either deny or allow all origins. Some specific instances are:

1. A resource that is not useful to applications from other origins, such as a login page, ~~ought~~SHOULD ~~not~~NOT to return an `Access-Control-Allow-Origin` ~~header~~response header. The resource still must protect itself against CSRF attacks~~, such as by requiring the inclusion of an unguessable token in the explicitly provided content of the request~~. The security properties of such resources are unaffected by user-agents conformant to this specification.
2. A ~~resource that is~~ publicly accessible resource ~~, with no access control checks,~~which is intended to uniformly process all incoming requests can ~~always safely~~be reasonably made available to any cross-origin requests. Such resources SHOULD return an `Access-Control-Allow-Origin` ~~header~~response header whose value is "`*`".
3. A `GET` response whose entity body happens to parse as ECMAScript ~~can~~MAY return an `Access-Control-Allow-Origin` ~~header~~response header whose value is "`*`" provided there are no sensitive comments, as the script content of such a resource~~it~~ can be accessed cross-origin, independently of CORS, using an HTML `script` element. If needed, such resources can implement access control and CSRF protections as described above.

4.3 Cross-origin Requests and User Credentials

Care must always be taken by applications when making cross-origin requests with user credentials. ~~, and servers processing such requests must take care in the use of credentials, including the `Origin` header.~~ In particular:

1. The `Origin` ~~header~~request header is intended to allow a server to grant read access to a returned resource representation, across origins, in the user agent context. Servers SHOULD use the value of the `Origin` ~~header~~request header to return a correct and minimally scoped Access-Control-Allow-Origin response header. Servers MAY, as a performance optimization, use the value of `Origin` ~~header~~request header to decline to calculate or return ~~the~~a resource representation ~~of a resource~~ when requested by a disallowed origin. Servers SHOULD NOT ~~When requests have significance other than retrieval, and when~~ rely~~ing~~ on the `Origin` ~~header~~request header as a credential~~, servers must be careful to distinguish between authorizing a request and authorizing access to the representation of that resource in the response~~ to authorize write or execute access to a resource.

Servers that do choose to ~~use~~rely on the `Origin` ~~header~~request header as a credential for authorizing write or execute access to a resource are encouraged to consider the following.

a) ~~Authorization for a request should be performed~~Servers SHOULD authorize requests using only the intersection of the authority of the user and the requesting origin(s). In the case of redirects, more than one value for `Origin` may be present and all must be authorized.

b) Servers using the `Origin` ~~header~~request header to authorize requests are encouraged to also verify that the `Host` ~~header~~request header matches its expected value to ~~prevent~~ requests being forward~~ing attacks~~ed by malicious servers. Consider two ~~sites~~servers, `corp.example` and `corp.invalid`. A web application ~~at~~ client instance from `corp.example` makes a cross-origin request to `corp.invalid`, and the user agent sends the `Origin` ~~header~~request header value "`corp.example`". If `corp.invalid` or the network is malicious, it may ~~ay~~ be able to cause the request to be instead delivered to the `corp.example server`, with the result that `corp.example` would receive a request that appears to originate from itself. Verifying the `Host` ~~header~~request header would reveal that the user agent intended the request for `corp.invalid` and it can be discarded. Even better would be to exclusively use secure connections for cross-origin requests to enable user agents to detect such misdirections.

c) Before honoring cross-origin requests with user credentials, i~~t~~t is often appropriate ~~for servers to require an authorization ceremony~~to ask~~ing~~ the~~a~~ user to consent ~~that cross-origin~~to such requests ~~with credentials be honored~~ from each ~~a~~ given origin. In such cases~~,~~, passing security tokens explicitly as part of the cross-origin request can remove any ambiguity as to the scope of authorization. Such user authorization ceremonies and authorization tokens of this sort are not part of this specification. OAuth ~~is~~ provides an example of this alternative pattern. [OAUTH]

2. Use of user credentials in a cross-origin request is appropriate when:

a) A cross-origin request with credentials as defined in this specification is used to substitute for alternate methods of authenticated resource sharing, such as server-to-server back channels, JSONP, or cross-document messaging. [JSONP][HTML]

This substitution can expose additional attack surface in some cases, as a cross-site scripting vulnerability in the requesting origin can allow elevation of privileges against the requested resource when compared to a server-to-server back channel.

Relative to~~As a substitute for~~ JSONP-style cross-origin credentialed requests, use of this specification significantly improves the security posture of the requesting applicatio~~nn,~~ as CORS~~it~~ provides cross-origin ~~data~~ access to data whereas JSONP operates via cross-origin code-injection. The requesting application ~~has to~~must still validate that data

received from origins that are not completely trusted conforms to expected formats and authorized values.

As a substitute for cross-origin communication techniques relying on loading a resource, with credentials, into an HTML `iframe` element, and subsequently employing cross-document messaging or other cross-origin side channels, this specification provides a roughly equivalent security posture. Again, data received from origins that are not completely trusted has to be validated to conform to expected formats and authorized values.

b) For resources that are safe and idempotent per HTTP, and where the credentials are used only to provide user-specific customization for otherwise publicly accessible information. In this case, restricting access to certain origins may protect user privacy by preventing customizations from being used to identify a user, except at authorized origins.

~~2.~~3. When this specification is used for requests which have significance other than retrieval and which involve coordination between or data originating from more than two origins, (e.g. between resources enabling editing, printing and storage, each at distinct origins) requests ~~ought to~~SHOULD set the omit credentials flag and servers ~~ought to~~SHOULD perform authorization using security tokens explicitly provided in the content of the request, especially if the origins are not all mutually and completely trusted.

In such multi-origin scenarios, a malicious resource at one of the origins may be able to enlist the user-agent as a confused deputy and elevate its privileges by abusing the user's ambient authority. Avoiding such attacks requires that the coordinating applications have explicit knowledge of the scope of privilege for each origin and that all parameters and instructions received are carefully validated at each step in the coordination to ensure that effects implied do not exceed the authority of the originating principal. [CONFUSED]

Given the difficulty of avoiding such vulnerabilities in multi-origin interactions it is recommended that, instead of using ~~implicit~~ the ambient authority of user credentials such as cookies, security tokens which specify the particular capabilities and resources authorized should be passed explicitly as part of each request. OAuth again provides an example of such a pattern.

4.4 Malicious Content

Authors of client-side Web applications are strongly encouraged to validate content retrieved from a cross-origin resource as it might be harmful.

4.5 Boundaries More Granular than An Origin

~~Authors of client-side~~ Web applications using~~-~~ boundaries more granular than an origin (for example identifying the security principal responsible for a resource by a~~a~~ URL of the type `people.example.org/~author-name/`~~-~~) ~~are~~ will not be able to securely utilize the mechanism in this specification. O~~to be aware that o~~nly cross-origin security is

provided by this and related specifications and ~~that~~ therefore using a distinct origin rather than distinct path is vital for secure client-side Web applications.