WEB420 Discussion 8.1

Describe what “Cache-Control” is and how it is used

Cache-Control is an HTTP header for both client requests and server responses that specifies such browser caching policies as how a resource is cashed and where and how long the cache will last before expiring[[1]](#footnote-1). These policies are called directives. Cache-Control has a number of directives, including, but not limited to, the following:

* Max-Age: defines in seconds how long it takes for a cached copy of a resource to expire, requiring the browser to send another request to the server
* No-Cache: means the browser can cache a response only after submitting a validation request to an origin server
* No-Store: browsers can’t cache the response, period
* Public: the resource can be cached by anyone
* Private: whether the resource can be cached depends on the user/client

The directives of Cache-Control are all delivered via the header in the response and apply to the entire HTTP response, not just to the body of the response.

List and describe the available options for managing security in a REST API

The first way to secure a RESTful API is to make sure it’s stateless, so it doesn’t depend on cookies or sessions but instead requires authentication every time a request is made[[2]](#footnote-2). After that, you have several choices for security. The least secure of those choices is Basic Authorization. Basic Auth simply depends on a username and password. Unfortunately, the password can easily be discovered. This can be solved by using HTTPS, which simplifies credentials to a randomly generated access token to be delivered in the username field. However, if a user is using several clients for a specific API, and a problem occurs on even one client which requires the user to change their credentials on that client, the user also has to change credentials on all other clients.

The next option is OAuth 1.0, where every client gets their own set of credentials. OAuth 1.0 allows you to integrate a website or app like Facebook with another site or app so that the user can authenticate through Facebook rather than directly through the other site or app. The other site requests temporary credentials from Facebook, the borrower is taken to Facebook where they either need to log in and allow Facebook to authorize credentials for the other site, and if the user does, they go back to the other site and can proceed. Unfortunately, this process can be rather disruptive if you’re using another app or something.

OAuth 2.0 tries to fix this by having the authenticating app such as Facebook redirect the user to a URL that has the access token, which the other website or application can then read without needing temporary credentials.

1. Cache Control (article). (no author). (n.d.). Retrieved from <https://www.imperva.com/learn/performance/cache-control/> [↑](#footnote-ref-1)
2. REST API Security Essentials (article) (no author). (n.d.). Retrieved from <https://restfulapi.net/security-essentials/> [↑](#footnote-ref-2)