**Delimiter Based Cache Deception:**

**Exploiting Delimiter Discrepancies:**

You may be able to use a delimiter discrepancy to add a static extension to the path that is viewed by the cache, but not the origin server. To do this, you'll need to identify a character that is used as a delimiter by the origin server but not the cache.

Firstly, find characters that are used as delimiters by the origin server. Start this process by adding an arbitrary string to the URL of your target endpoint. For example, modify /settings/users/list to /settings/users/listaaa. You'll use this response as a reference when you start testing delimiter characters.

**\*\*\*\*Note:**

**If the response is identical to the original response, this indicates that the request is being redirected. You'll need to choose a different endpoint to test. ESSENTIALLY IF WE DON’T GET 404 OR RETURNED TO A DIFFERENT PAGE ITS NOT VULNERABLE\*\*\*\***

Next, add a possible delimiter character between the original path and the arbitrary string, for example /settings/users/list;aaa:

* If the response is identical to the base response, this indicates that the ; character is used as a delimiter and the origin server interprets the path as /settings/users/list.
* If it matches the response to the path with the arbitrary string, this indicates that the ; character isn't used as a delimiter and the origin server interprets the path as /settings/users/list;aaa.

Once you've identified delimiters that are used by the origin server, test whether they're also used by the cache. To do this, add a static extension to the end of the path. If the response is cached, this indicates:

* That the cache doesn't use the delimiter and interprets the full URL path with the static extension.
* That there is a cache rule to store responses for requests ending in .js.

Make sure to test all ASCII characters and a range of common extensions, including .css, .ico, and .exe. We've provided a list of potential delimiter characters to get you started in the labs, see the [Web cache deception lab delimiter list](https://portswigger.net/web-security/web-cache-deception/wcd-lab-delimiter-list). **\*\*\*\* These are all accounted for in the updated web cache scanner payloads \*\*\*\***

You can then construct an exploit that triggers the static extension cache rule. For example, consider the payload /settings/users/list;aaa.js. The origin server uses ; as a delimiter:

* The cache interprets the path as: /settings/users/list;aaa.js
* The origin server interprets the path as: /settings/users/list

The origin server returns the dynamic profile information, which is stored in the cache.

Because delimiters are generally used consistently within each server, you can often use this attack on many different endpoints.

**\*\* The best way to approach delimiters testing is just to fuzz with all delimiters and file extensions (especially all delimiters). Sometimes, once we get a valid delimiter and file extension the response will then show extra cache headers explicitly showing us the response was in fact cached. Once again this exploit RELIES on discrepancies between the caching server/service and the backend server. This means we should be thoroughly testing this even on the most hardend targets. It is easy for devs to miss this discrepancy and introduce a bug. Use the web cache scanner, and also manually try to test for the “reverse method” basically doing: “/assets/test.js<delim-here>../../sensitivePage” (discussed later on in this doc). Furthermore, we can perform initial manual testing by adding arbitrary chars to an endpoint we want to test. If we get the same response even with adding the chars the page is not vulerbale. For example if were testing /profile/123 we can try /profile/123aaa if we get the same exact response with both this endpoint is not vulnerable to delim based deception (NOTE: still try the path mapping discrepancies mentioned below)\*\***

**Delimiter decoding discrepancies – full explaination**

Websites sometimes need to send data in the URL that contains characters that have a special meaning within URLs, such as delimiters. To ensure these characters are interpreted as data, they are usually encoded. However, some parsers decode certain characters before processing the URL. If a delimiter character is decoded, it may then be treated as a delimiter, truncating the URL path.

Differences in which delimiter characters are decoded by the cache and origin server can result in discrepancies in how they interpret the URL path, even if they both use the same characters as delimiters. Consider the example /profile%23wcd.css, which uses the URL-encoded # character:

* The origin server decodes %23 to #. It uses # as a delimiter, so it interprets the path as /profile and returns profile information.
* The cache also uses the # character as a delimiter, but doesn't decode %23. It interprets the path as /profile%23wcd.css. If there is a cache rule for the .css extension it will store the response.

In addition, some cache servers may decode the URL and then forward the request with the decoded characters. Others first apply cache rules based on the encoded URL, then decode the URL and forward it to the next server. These behaviors can also result in discrepancies in the way cache and origin server interpret the URL path. Consider the example /myaccount%3fwcd.css:

* The cache server applies the cache rules based on the encoded path /myaccount%3fwcd.css and decides to store the response as there is a cache rule for the .css extension. It then decodes %3f to ? and forwards the rewritten request to the origin server.
* The origin server receives the request /myaccount?wcd.css. It uses the ? character as a delimiter, so it interprets the path as /myaccount.

EXPLOITING Delim Decoding discrepancies:

you may be able to exploit a decoding discrepancy by using an encoded delimiter to add a static extension to the path that is viewed by the cache, but not the origin server.

Use the same testing methodology you used to identify and exploit delimiter discrepancies, but use a range of encoded characters. Make sure that you also test encoded non-printable characters, particularly %00, %0A and %09. If these characters are decoded they can also truncate the URL path.

\*\* THESE ARE ALSO ACCOUNTED FOR IN THE WEB CACHE SCANNER PAYLOADS \*\*