**Constructing a web cache poisoning attack**

Generally speaking, constructing a basic web cache poisoning attack involves the following steps:

1. [Identify and evaluate unkeyed inputs](https://portswigger.net/web-security/web-cache-poisoning#identify-and-evaluate-unkeyed-inputs)
2. [Elicit a harmful response from the back-end server](https://portswigger.net/web-security/web-cache-poisoning#elicit-a-harmful-response-from-the-back-end-server)
3. [Get the response cached](https://portswigger.net/web-security/web-cache-poisoning#get-the-response-cached)

Try to look for signs of a cache by way of headers or technologies (even if these signs are not present there is very likely a cache in place).

Then run param miner on requests to find unkeyed and potentially reflected inputs.

Finally, we have to work to get this harmful response cached which is half of the battle by itself. This will usually require playing around with the payload for a while. Getting the response cached depends on many factors like file extension, content type, route, status code and response headers. The best thing is to spend time learning how the cache behaves. Look for headers that indicate the response was cached or study the time it takes for a response to render. Short load times indicate the response was cached. Look for headers like Age, or Cache-Control or any other header that includes the word cache. Even headers like Via could have a value of some technology that indicates the use of a cache. If you see random technologies being mentioned in the response headers look them up!

**Exploits:**

XSS through cache poisoning:

To pull this off we must find headers that are reflected into the response body. This can of course come in the form of many different headers. Often we will see it in things like x-forwarded-host or host or location even as these are commonly used to generate dynamic paths or URLs to then be added to the body.

Often we can see cookies being reflected in the responses as well. This could also be an attack surface.

----------------------

Malicious Redirects:

This normally happens on requests to static files that rely on dynamic URLs or paths from designated headers. Changing these headers could lead to users being redirected to our own website enabling us to run JS on them. This is often seen with things like the Host header or X-Forwarded-Host etc. Sometimes these headers will only take a path. However, sometimes other headers can be used to add https to the path. This could be something like X-Forwarded-Scheme for example. The redirect must use https to redirect to our own server, if not then its simply a local path.

-------------------------

Cache Poisoning to Exploit DOM-based Vulnerabilities:

Many sites use JS to fetch and process data from the back-end if we can pass in some input to this JS and get it cached, it will be displayed to other users. For example if the websites fetchs some JSON file and uses that data in some javascript elsewhere. If we can host our own malicious JSON or simply change the JSON in some way, this could get cached and be used as input to the JS running on other users. This could just be JS injection or it could require hosting our own JSON data and poisoning another resource thus making it fetch our malicious JSON. \*\*\* we will likely need to add access-control-allow-origin: \* to our hosts site somehow. This will allow the target site to retrieve our data. Just consider CORS if we are running into issues here \*\*\*