SCENARIO

The application is vulnerable to web cache poisoning because it accepts GET requests that have a body, but does not include the body in the cache key. There is also inconsistent parameter parsing between the cache and the back-end. We will try to poison the cache with a response that executes alert(1) in the visitor's browser.

**PROCEDURE**

1. Open the web application and send the request for **/js/geolocate.js?callback=setCountryCookie** to BurpSuite’s Repeater and start testing it by adding Cache Buster to the URL.
2. Observe that every page imports the script **/js/geolocate.js**, executing the callback function **setCountryCookie()**.
3. Notice that we can control the name of the function that is called in the response by passing in a duplicate callback parameter via the request body. Also notice that the cache key is still derived from the original callback parameter in the request line.
4. According to the way our injected query parameter we will craft an exploit string which will break out of that tag and trigger our alert by executing arbitrary JavaScript.
5. Append the crafted exploit as shown in the Payload in the request body.
6. Send the malicious request after removing the cache buster parameter and keep replaying the request until we see our exploit server URL being reflected in the response and **X-Cache: hit** in the headers.

**PAYLOAD**

callback=alert(1)

**PROOF OF CONCEPT**

****

**REMEDIATION**

1. **GET Request Norms:** Adhere to HTTP protocol specifications. The HTTP specification doesn't forbid sending a body with a GET request, but it's unusual and most applications don't support it. It's best practice to avoid sending bodies with GET requests to prevent potential issues like this.
2. **Consistent Parsing:** Ensure that both the cache and the backend parse parameters consistently. The cache key generation and the backend should both consider all parts of the request, including unusual parts like a body with a GET request.
3. **Strict Input Validation:** Apply strict validation rules for the input. Reject any requests that don't adhere to expected formats, especially for parameters that can be executed as code.
4. **Body in Cache Key:** If, for any reason, you have to support bodies in GET requests, make sure to include the body content in the cache key.
5. **Avoid Reflecting Input:** If possible, avoid reflecting user input in the response, especially in locations where it can be executed as code.
6. **Sanitize Input:** Ensure that all input, especially those that can be reflected, is sanitized to prevent potential script injections. Escape characters that have special meanings in JavaScript or HTML.
7. **Cache Key Inclusion:** All components of a request, including headers, parameters, and body, should be included in the determination of the cache key to prevent poisoning.