SCENARIO

The application is vulnerable to web cache poisoning due to discrepancies in how the cache and the back-end application handle ambiguous requests. 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 we can use cache buster if we append **?cb=121** into the URL we can query the server.
3. Notice that we can add a second Host header with an arbitrary value, this appears to be ignored when validating and routing your request. Crucially, notice that the arbitrary value of your second Host header is reflected in an absolute URL.
4. Go to the exploit server and create a file at Payload 1 containing the Payload 2 in the body. Store the exploit.
5. Back in Burp Repeater, remove the cache buster and try to mimic the request by replaying the request in browser.
6. Send the malicious request 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. /resources/js/tracking.js
2. alert(document.cookie)

**REMEDIATION**

1. **Consistent Handling of Ambiguous Requests:** It's crucial to ensure that both the caching layer and the backend application handle requests consistently. Discrepancies, like how they interpret headers, can lead to vulnerabilities like cache poisoning.
2. **Whitelist of Accepted Headers:** Only allow specific, known headers to be processed. If a header isn't on the whitelist, it should be ignored or the request should be rejected.
3. **Strip Unnecessary Headers:** Before processing a request, remove any headers that aren't explicitly required. This minimizes the attack surface and ensures that headers like a duplicate Host header don't introduce vulnerabilities.
4. **Validate and Sanitize All Input:** All inputs, including headers and query parameters, should be rigorously validated against a set of strict rules. Anything that doesn't meet the criteria should be rejected. Additionally, sanitize inputs to remove or neutralize potentially malicious content.
5. **Consistent Cache Key Generation:** Ensure that cache keys are generated consistently, taking into account all parts of a request that might affect the response. This includes headers, query parameters, and other factors.
6. **Implement Cache Control Headers:** Use cache control headers, like no-cache and no-store, to prevent or control the caching of sensitive responses. This ensures that malicious content isn't inadvertently stored and served to users.