SCENARIO

The application is vulnerable to web cache poisoning that is only exploitable when you use multiple headers to craft a malicious request. We will try to poison the cache with a response that executes alert(document.cookie) in the visitor's browser.

**PROCEDURE**

1. Open the web application and in the BurpSuite’s Proxy tab send the GET request for the JavaScript file /resources/js/tracking.jsrequest to BurpSuite’s Repeater and study it.
2. Try adding a **cache buster and X-Forwarded-Host** parameters in the request and we see that this doesn’t work.
3. Now, remove the **X-Forwarded-Host** header and add the **X-Forwarded-Scheme** header instead. Notice that if we include any value other than HTTPS, we receive a 302 response. The **Location** header shows that we are being redirected to the same URL that we requested, but using **https** this time.
4. Add the **X-Forwarded-Host: example.com** header back to the request, but keep **X-Forwarded-Scheme: nothttps** as well. Send this request and notice that the Location header of the 302 redirect now points to <https://example.com/>.
5. Go to the exploit server and replace the name with the Payload 1 and the body with the Payload 2 and store the exploit now.
6. Get back to BurpSuite’s Repeater and put the value of the **X-Forwarded-Host** parameter as our exploit server URL and the **X-Forwarded-Scheme** as anything other than HTTPS.
7. Send the request until the exploit server URL is reflected in the response and **X-Cache: hit** in the headers.
8. Send the malicious request after removing the cache buster parameter from the URL 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. **Ignore Headers from Untrusted Sources:** Ensure that your application and caching solutions only consider the headers that are necessary for your application logic and ignore headers like X-Forwarded-\* unless they are strictly required and come from trusted sources.
2. **Consistent Caching Behavior:** Ensure consistent behavior between caching mechanisms and the application. They should both handle headers and other inputs the same way.
3. **No Cache for Dynamic Responses:** Avoid caching responses that can change based on varying inputs, especially when the input can be controlled by users, such as the headers in this case.
4. **Validation and Whitelisting:** Only allow known, safe values for headers and parameters that can affect application logic or response content. For example, only accept https in the X-Forwarded-Scheme header.
5. **Use Application Firewalls:** Web Application Firewalls (WAF) can be configured to block or alert on unusual headers or combinations of headers.
6. **Limit Scope of Cache:** If caching is absolutely required, limit the scope. For example, don't cache responses with certain suspicious headers.