SCENARIO

The application is vulnerable to web cache poisoning because the query string is unkeyed. We will try to poison the cache with a response that executes alert(1) in the visitor's browser.

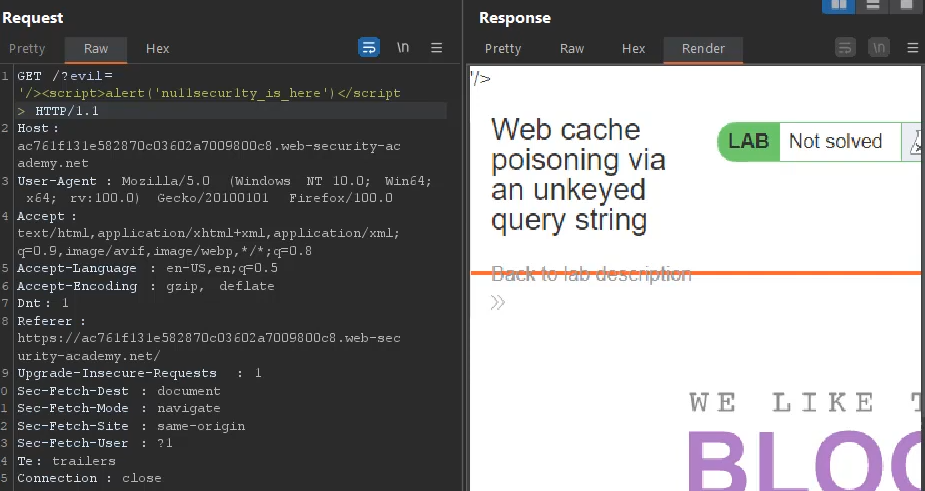
**PROCEDURE**

1. Open the web application and in the BurpSuite’s Proxy tab send the GET request for homepage to BurpSuite’s Repeater and study it.
2. In order to display the cache key in the response we will include the Payload 1 in the request.
3. Try adding any arbitrary query parameters to the request and we see that it does not matter as we still receive the cached response because they are not included in the cache key generation.
4. Try adding Payload 2 in the request and we see that we can successfully use that parameter as a cache buster.
5. When we get a cache miss, notice that our injected parameters are reflected in the response. If the response to our request is cached, we can remove the query parameters and they will still be reflected in the cached response.
6. 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.
7. 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. Pragma: x-get-cache-key
2. Origin: [www.hacker.com](http://www.hacker.com)
3. ?hacked='/><script>alert(1)</script>

**PROOF OF CONCEPT**

****

**REMEDIATION**

1. **Validate Input:** All input, including query parameters, should be rigorously validated. Only expected parameters should be accepted, and unexpected ones should be ignored.
2. **Avoid Dynamic Content Based on Unvalidated Inputs:** If content is dynamically created based on user input, that input should be sanitized and validated. Dangerous characters like <> and other code characters should be filtered or properly encoded.
3. **Use Comprehensive Cache Keys:** The cache key should include all parts of the request that can influence the response, especially if these can be manipulated by the client. If a caching system allows certain parameters to bypass the cache or force a cache miss, they should be used responsibly and with understanding.
4. **Limit Caching:** Highly dynamic content, especially if it's influenced by user input, should not be cached. If it must be cached, mechanisms should be in place to ensure the safety of the cache's contents.
5. **Configure Cache Headers Properly:** Ensure that cache headers such as Cache-Control are set appropriately. Utilize directives like no-store and no-cache for sensitive or dynamic content.
6. **Properly Implement Origin Headers:** Validate the Origin header to prevent misuse. Only accept known origins and reject or ignore unknown or unexpected ones.