SCENARIO

The application is vulnerable to web cache poisoning because it excludes a certain parameter from the cache key. 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. Try using a cache buster in the URL and we see that it works and we see the query string reflected in the response.
3. Use Param Miner's **"Guess GET parameters"** feature to identify that the parameter **utm\_content** is supported by the application.
4. Notice that this parameter is unkeyed and we get a cache hit everytime even if we change the value.
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. Append the crafted exploit as shown in the Payload in the URL.
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**

?utm\_content='/><script>alert(1)</script>

**PROOF OF CONCEPT**

**REMEDIATION**

1. **Ensure Comprehensive Cache Key Generation:** Every aspect of the request that can influence the response should be incorporated into the cache key. If certain parameters are deliberately excluded from cache keys, understand and document the reason.
2. **Input Validation:** Ensure that all user-supplied data, including the utm\_content parameter and other potential tracking parameters, are rigorously validated. If they are reflected in content, ensure they are sanitized to prevent script injection.
3. **Limit Cacheable Content:** Consider limiting what content can be cached. For instance, content that reflects user input directly might not be suitable for caching.
4. **Content Sanitization:** Even if content is going to be cached, sanitize it to prevent potential script injections. Escape characters that have special meanings in HTML, like <>, and avoid directly embedding user input in scripts or styles.
5. **Use Safe Defaults:** When caching, the default behaviour should be safe. If a parameter isn't recognized, the safe option is not to cache the result rather than omitting the unrecognized parameter from the cache key.