SCENARIO

The application is vulnerable to web cache poisoning that is not directly exploitable due to browser URL-encoding. We will try to take advantage of the cache's normalization process to exploit this vulnerability. Find the XSS vulnerability and inject a payload that will execute alert(1) in the victim's browser.

**PROCEDURE**

1. Open the web application and send the request for **homepage** to BurpSuite’s Repeater.
2. Try navigating to a non-existent URL by appending Payload 1 into the URL.
3. We see that our Payload is included in the response of the 404 not found page.
4. 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.
5. Notice that if we request this URL in the browser, the payload doesn't execute because it is URL-encoded the next time we execute the same payload.
6. Append the crafted exploit as shown in the Payload in the request body.
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. /hacker
2. /hacker</p><script>alert(1)</script><p>Hacked!

**PROOF OF CONCEPT**

****

**REMEDIATION**

1. **Normalize Inputs Consistently:** Ensure that the cache and the backend handle and normalize URLs and parameters consistently. The backend should decode and sanitize inputs in the same way that the cache does.
2. **Avoid Reflecting Inputs in Responses:** As a general rule, avoid reflecting user inputs in responses, especially in error pages. This reduces the attack surface for potential script injection.
3. **Sanitize Error Pages:** If user inputs must be reflected in error pages or other responses, sanitize the input rigorously to prevent potential code execution. This includes stripping or neutralizing special characters and tags that can be interpreted as code.
4. **Educate Developers:** Make sure that developers understand the potential risks associated with reflected user input and the importance of sanitizing all data that's reflected in responses.
5. **Implement Content Security Policy (CSP):** Use a strict CSP to reduce the risk of cross-site scripting attacks. A CSP can prevent inline scripts from running and restrict scripts to trusted sources.