SCENARIO

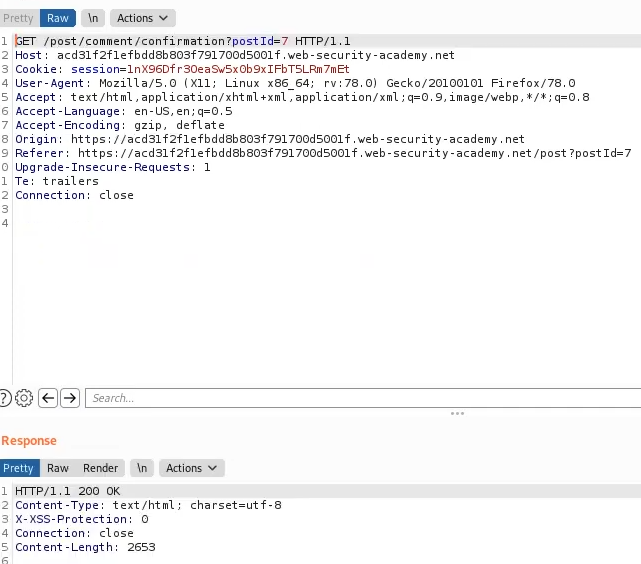
The application contains a stored cross site scripting vulnerability in the comment box functionality. We will try to trigger an alert message by injecting a payload into the search box.

**PROCEDURE**

1. Go to the vulnerable application and comment something with all the fields filled.
2. Check out the response/source and you’ll find that the application puts the website into a href HTML tag unsafely.
3. We don’t even need to break out of that tag because we can simply execute JavaScript into that tag by using the given payload.

**PAYLOAD**

javascript:alert(“HACKED!”)

**PROOF OF CONCEPT**

**REMEDIATION**

1. **Output Encoding:** Ensure all dynamic data is appropriately escaped for the context it will be used in. When inserting dynamic data into HTML, use appropriate methods/functions to prevent HTML or JavaScript injection. For instance, instead of using traditional string concatenation, utilize functions that safely encode strings for insertion into HTML.
2. **Whitelist Input Validation:** Rather than trying to identify and block potentially malicious input, define what constitutes valid input and reject anything outside of that criteria. For instance, if a comment box should only accept alphanumeric characters, then filter out anything that doesn't fit that criteria.
3. **Use Safe APIs:** Use frameworks and APIs that are known to be safe and automatically escape user input, such as innerText (instead of innerHtml) in JavaScript, or parameterized SQL queries to prevent SQL injection.