**Lab 14: Cross-site scripting - Reflected**

**Objective**:

* This lab is dedicated to exploring Reflected Cross-Site Scripting (XSS) vulnerabilities, where malicious scripts are injected into a web application and reflected back to the user's browser. Participants will interact with a simulated web application that is vulnerable to reflected XSS attacks. The objective is to understand how to identify and exploit these vulnerabilities, as well as to learn effective mitigation techniques to protect web applications from such attacks.

In this lab, students need to:

* Answer the following questions:
  + What is Reflected Cross-Site Scripting (XSS), and how does it differ from other types of XSS attacks like Stored XSS and DOM-Based XSS?
  + Explain how Reflected XSS works, including how malicious scripts are injected and executed in the victim's browser, and discuss the typical attack vectors for this type of vulnerability.
* Perform challenge:
  + [Reflected DOM XSS](https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-dom-xss-reflected)
* Explain and capture all steps (full windows screen capture).

Submit a report addressing all the questions mentioned above in either **PDF** or **Markdown** format. Additionally, include a **video** demonstrating the detailed process of your work to ensure the authenticity of your lab exercise.

**Lab 14: Cross-site scripting - Reflected**

**What is Reflected Cross-Site Scripting (XSS), and how does it differ from other types of XSS attacks like Stored XSS and DOM-Based XSS?**

Reflected Cross-Site Scripting (XSS) is a type of security vulnerability commonly found in web applications. In a reflected XSS attack, malicious scripts are injected into web pages, often through input fields or URLs, and then reflected back to the user by the web server. When the user interacts with the compromised web page, the injected script executes within their browser, allowing the attacker to steal sensitive information, manipulate the page content, or perform other malicious actions.

The main difference between reflected XSS and other types of XSS attacks, such as Stored XSS and DOM-Based XSS, lies in how the malicious payload is injected and executed:

1. \*\*Reflected XSS\*\*: In reflected XSS attacks, the injected script is reflected back to the user by the web server without being stored on the server-side. This typically occurs when user input is not properly validated or sanitized by the application before being returned to the user. The malicious script is embedded within the URL or form submission, and when the user interacts with the compromised page, the script executes in their browser.

2. \*\*Stored XSS\*\*: Stored XSS occurs when the malicious script is permanently stored on the server-side, such as in a database, and is then served to multiple users when they request a particular page or resource. This type of XSS attack is particularly dangerous because it can affect multiple users and persist over time, leading to widespread exploitation if not properly mitigated.

3. \*\*DOM-Based XSS\*\*: DOM-Based XSS attacks exploit vulnerabilities in the Document Object Model (DOM) of a web page. Instead of targeting the server-side code, these attacks manipulate the client-side DOM directly. The malicious script is typically injected into the page's HTML or JavaScript and executed within the client's browser. Unlike reflected and stored XSS, which involve interactions with the server, DOM-Based XSS attacks occur entirely on the client-side.

In summary, while all types of XSS attacks involve injecting and executing malicious scripts within web applications, they differ in how the payload is injected, stored, and executed. Understanding these differences is crucial for effectively mitigating XSS vulnerabilities and protecting web applications from exploitation.

**Explain how Reflected XSS works, including how malicious scripts are injected and executed in the victim's browser, and discuss the typical attack vectors for this type of vulnerability.**

Reflected Cross-Site Scripting (XSS) works by injecting malicious scripts into web pages and reflecting them back to the user's browser from the server's response. Here's how it typically works:

1. \*\*Injection Point\*\*: The attacker identifies a vulnerable input field or parameter within a web application. This could be a search box, a comment form, a URL parameter, or any other input field where user-supplied data is not properly validated or sanitized by the application before being included in the server's response.

2. \*\*Payload Injection\*\*: The attacker then crafts a malicious payload, typically JavaScript code, and injects it into the vulnerable input field. For example, they might input something like `<script>alert('XSS')</script>` into a search box.

3. \*\*Server Reflection\*\*: When the user submits the form or makes a request containing the injected payload, the server processes the input and includes the payload in the response without proper validation or sanitization. As a result, the malicious script is reflected back to the user's browser as part of the server's response.

4. \*\*Execution in Victim's Browser\*\*: Upon receiving the server's response, the victim's browser interprets and executes the injected script within the context of the web page. This allows the attacker to achieve various malicious goals, such as stealing session cookies, redirecting the user to a phishing page, defacing the website, or performing actions on behalf of the user.

Attack vectors for Reflected XSS vulnerabilities include:

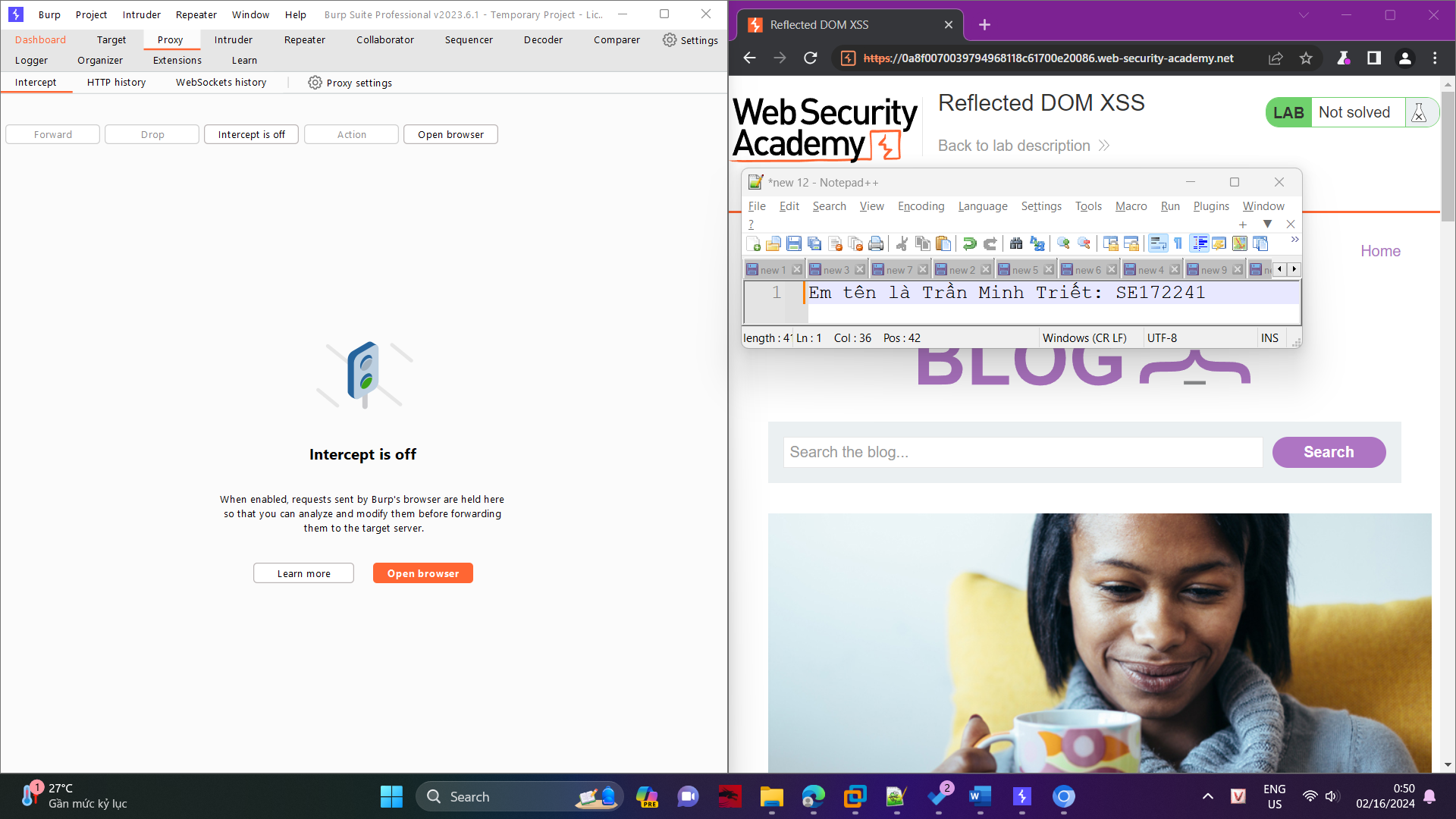
- \*\*URL Parameters\*\*: Attackers may craft malicious URLs containing the payload and trick users into clicking on them. For example, they might send a phishing email with a link to a legitimate-looking website that includes the malicious payload in the URL.

- \*\*Form Inputs\*\*: Input fields such as search boxes, login forms, and comment sections are common targets for XSS attacks. Attackers can inject the payload directly into these fields and submit the form to trigger the vulnerability.

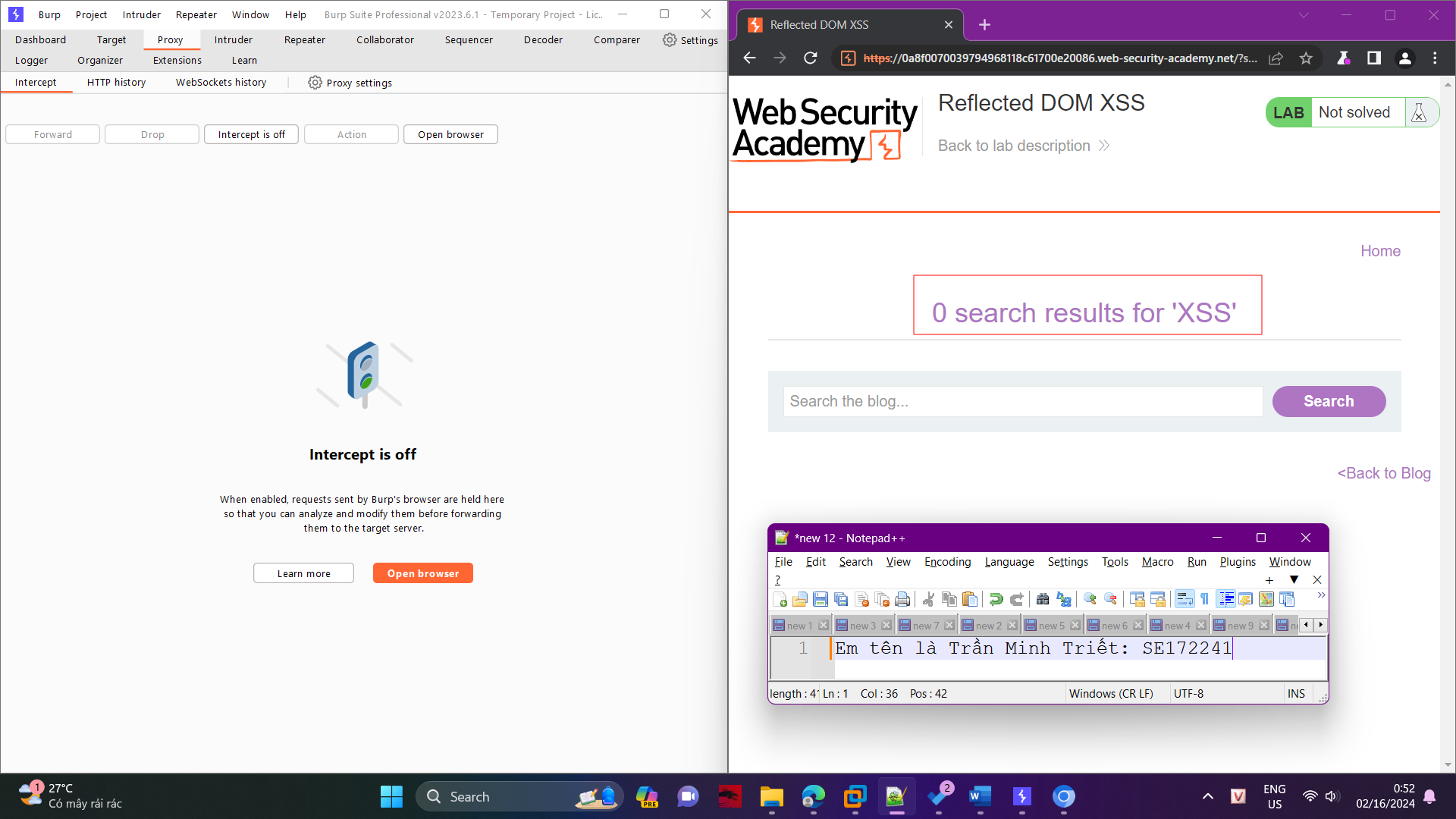
- \*\*HTTP Headers\*\*: Certain HTTP headers, such as the `Referer` header, can also be manipulated to inject XSS payloads. For instance, an attacker might craft a URL with a malicious `Referer` header that includes the payload, leading to a reflected XSS vulnerability when the victim visits the target website.

Overall, Reflected XSS vulnerabilities can have serious consequences and should be mitigated by implementing proper input validation, output encoding, and other security measures in web applications.

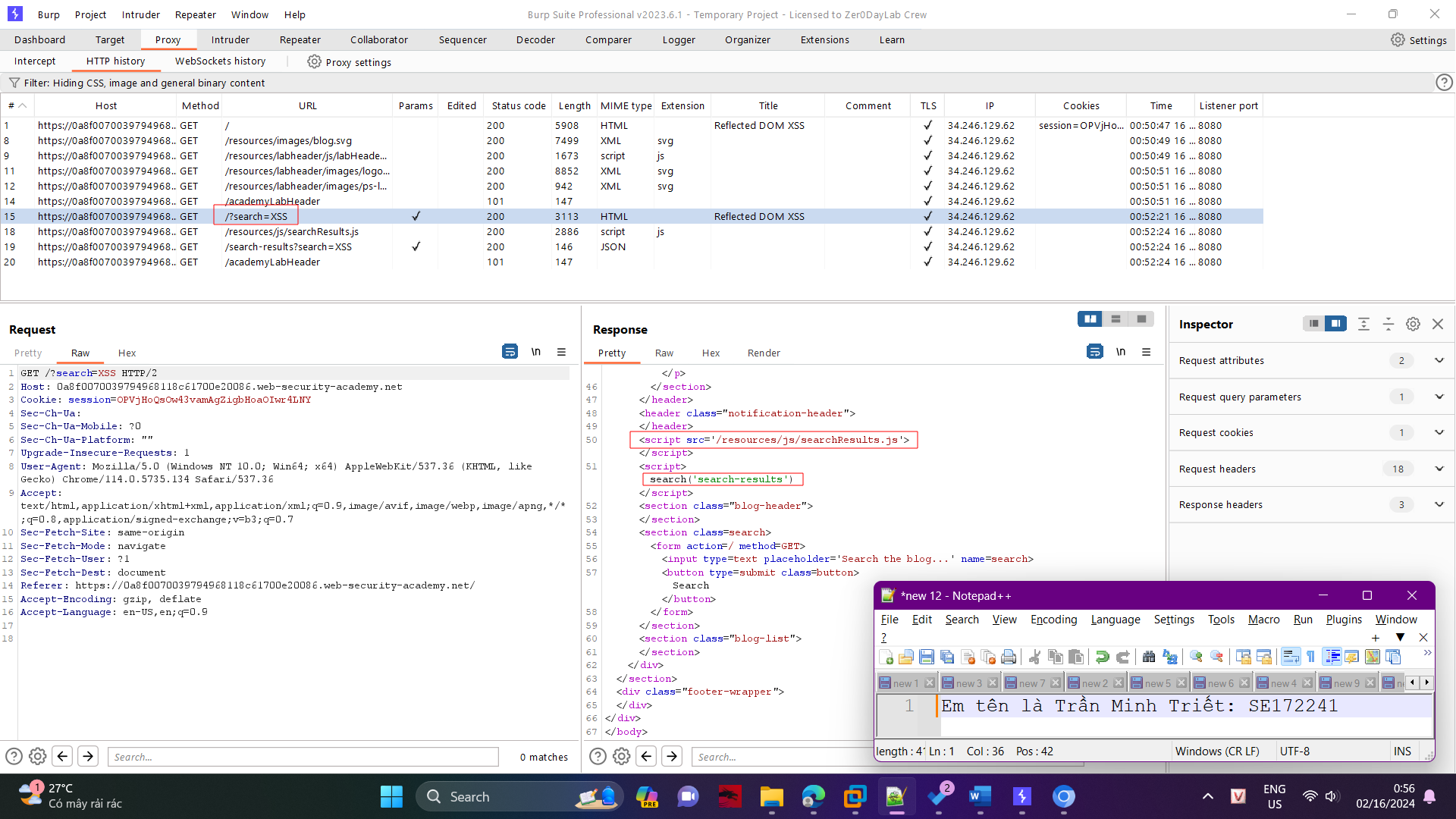
**Challenge**

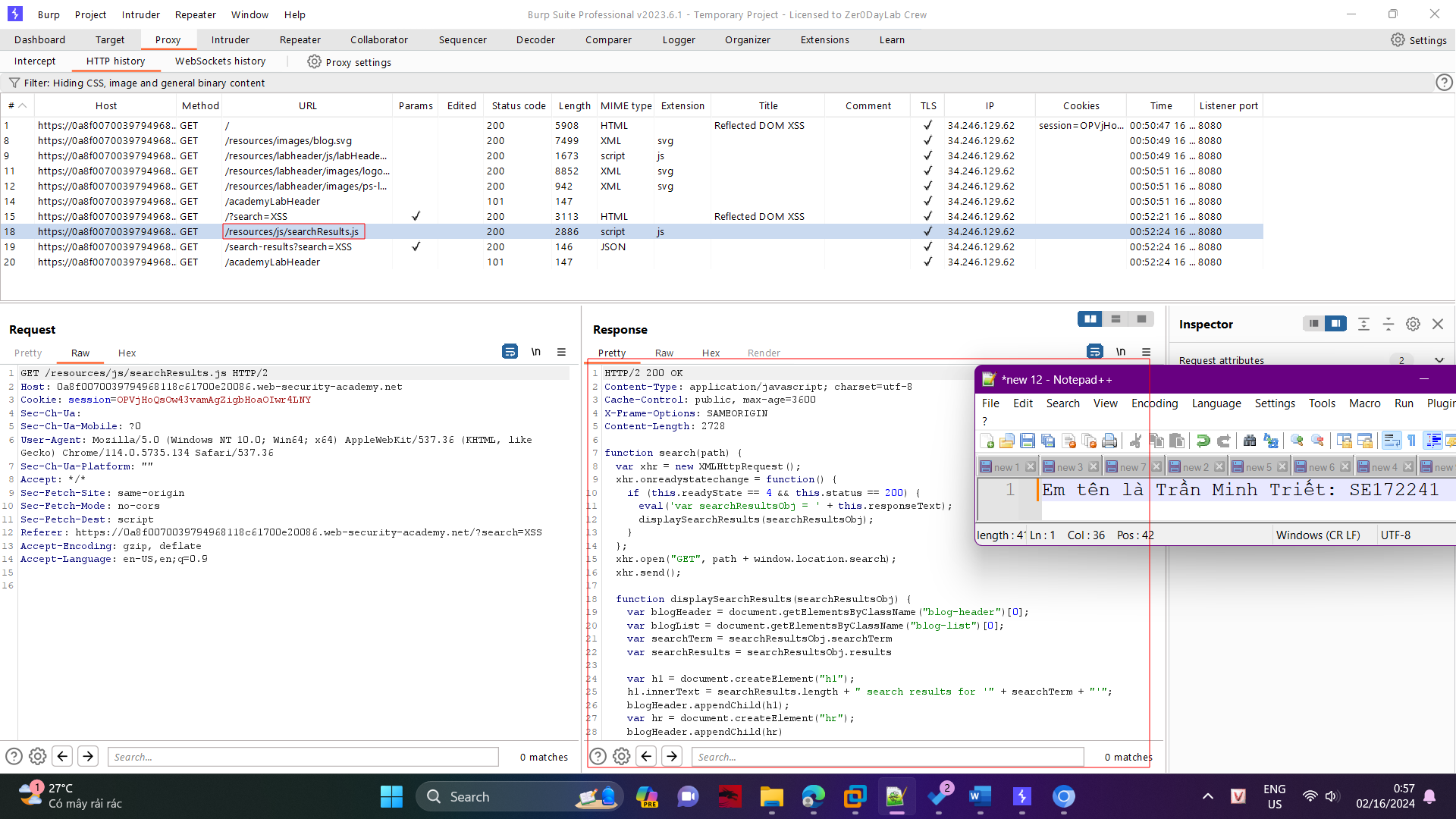


Back in the lab, go to the target website and use the search bar to search for a random test string, such as "XSS".

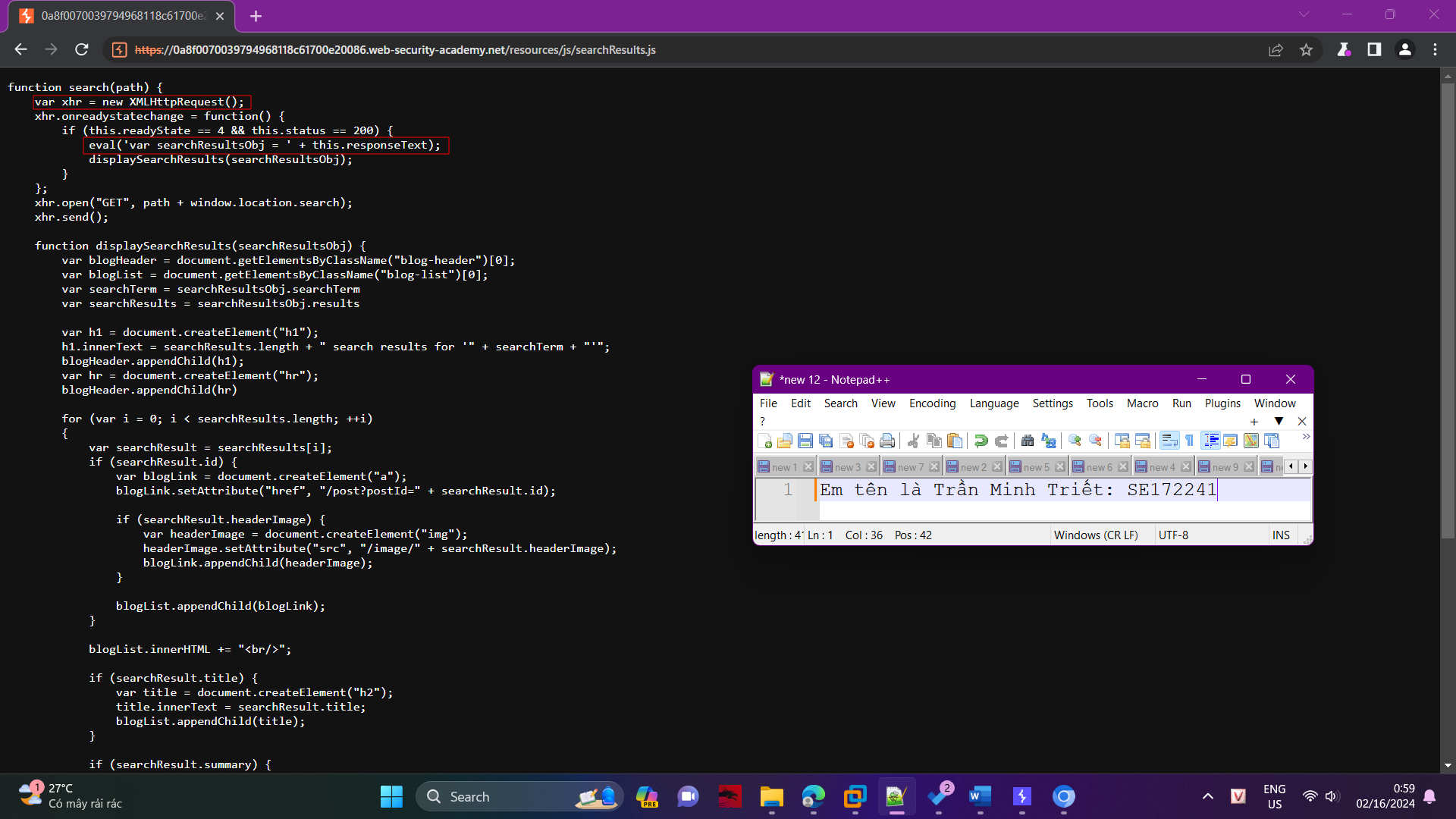


Return to the Proxy tool in Burp Suite and forward the request.





From the Site Map, open the searchResults.js file and notice that the JSON response is used with an eval() function call.



By experimenting with different search strings, you can identify that the JSON response is escaping quotation marks. However, backslash is not being escaped.

