**Lab 16: Server-side request forgery (SSRF)**

**Objective**:

* This lab centers around Server-Side Request Forgery (SSRF) attacks, where an attacker abuses the functionality of a server, causing it to access or manipulate information in unintended ways. Participants will engage with a simulated environment where an application is vulnerable to SSRF.

In this lab, students need to:

* Answer the following questions:
  + What is Server-Side Request Forgery (SSRF), and how does it differ from other types of web application vulnerabilities?
  + Describe a methodology for exploiting an SSRF vulnerability in a web application. What are the key steps and techniques involved in crafting and sending malicious requests to exploit SSRF vulnerabilities?
* Perform challenge:
  + [Blind SSRF with out-of-band detection](https://portswigger.net/web-security/ssrf/blind/lab-out-of-band-detection)
* 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 16: Server-side request forgery (SSRF)**

**What is Server-Side Request Forgery (SSRF), and how does it differ from other types of web application vulnerabilities?**

Server-Side Request Forgery (SSRF) is a web application vulnerability that allows an attacker to manipulate the server into making unintended requests to internal or external resources. This is typically achieved by tricking the server into making requests on behalf of the attacker, potentially leading to unauthorized access to sensitive data, server-side exploitation, or facilitating attacks against other systems.

Here's how SSRF differs from other types of web application vulnerabilities:

1. \*\*Cross-Site Scripting (XSS)\*\*: XSS involves injecting malicious scripts into web applications, which are then executed within the context of the victim's browser. In contrast, SSRF involves manipulating the server into making requests to other systems, potentially bypassing network security measures and accessing internal resources.

2. \*\*SQL Injection (SQLi)\*\*: SQL Injection attacks exploit vulnerabilities in web applications that use SQL databases by injecting malicious SQL queries. This can lead to unauthorized access to the database, data leakage, or data manipulation. SSRF, on the other hand, focuses on manipulating the server into making requests to other systems rather than directly interacting with the database.

3. \*\*Remote Code Execution (RCE)\*\*: Remote Code Execution vulnerabilities allow attackers to execute arbitrary code on a server, often leading to full compromise of the system. While SSRF may facilitate certain types of RCE attacks by allowing the attacker to make requests to internal services or APIs, SSRF itself does not necessarily involve executing arbitrary code on the server.

4. \*\*Cross-Site Request Forgery (CSRF)\*\*: CSRF attacks involve tricking authenticated users into performing unintended actions on a web application. This is typically achieved by exploiting the trust relationship between the user and the application. In contrast, SSRF does not target users directly but instead manipulates the server into making requests to other systems, potentially bypassing access controls or security measures.

5. \*\*Path Traversal\*\*: Path Traversal vulnerabilities allow attackers to access files or directories on a server that are not intended to be accessible. While SSRF may facilitate access to internal resources, it does not necessarily involve accessing files or directories directly on the server.

In summary, SSRF differs from other types of web application vulnerabilities in that it focuses on manipulating the server into making unintended requests to internal or external resources, rather than directly exploiting flaws in the application's code or user interactions. It can be used by attackers to bypass network security measures, access sensitive data, or facilitate attacks against other systems.

**Describe a methodology for exploiting an SSRF vulnerability in a web application. What are the key steps and techniques involved in crafting and sending malicious requests to exploit SSRF vulnerabilities?**

Exploiting a Server-Side Request Forgery (SSRF) vulnerability in a web application typically involves a series of steps to craft and send malicious requests to manipulate the server into making unintended requests to internal or external resources. Here's a methodology for exploiting an SSRF vulnerability:

1. \*\*Identify SSRF Vulnerability\*\*: The first step is to identify the presence of an SSRF vulnerability within the web application. This may involve manual testing, automated scanning, or analyzing the application's source code to find potential input points where the attacker can control the URL or destination of the server's requests.

2. \*\*Locate Internal Resources\*\*: If the application allows the server to access internal resources, such as URLs, files, or services, the attacker needs to identify these resources. This could include internal network services, metadata services, or other components accessible from the server's environment.

3. \*\*Craft Malicious Request\*\*: Once the attacker has identified the SSRF vulnerability and the internal resources they want to access, they craft a malicious request to exploit the vulnerability. This request may include manipulating the URL, HTTP headers, or other parameters to direct the server to the desired target.

4. \*\*Exploit Internal Services\*\*: The attacker can leverage the SSRF vulnerability to interact with internal services or APIs that are not intended to be directly accessible from outside the server's environment. This could involve querying metadata services, accessing sensitive files, or even exploiting vulnerabilities in internal systems.

5. \*\*Exploit External Services\*\*: In addition to accessing internal resources, SSRF vulnerabilities can also be used to interact with external services or APIs. Attackers may use the server's trusted network access to scan or attack external systems, bypassing network security measures such as firewalls or access controls.

6. \*\*Data Exfiltration\*\*: Once the attacker has successfully exploited the SSRF vulnerability to access internal or external resources, they may exfiltrate sensitive data back to their own server. This could include stealing credentials, extracting sensitive files, or compromising additional systems within the network.

7. \*\*Cover Tracks\*\*: To avoid detection, the attacker may attempt to cover their tracks by obfuscating their malicious requests, deleting logs or evidence of their activities, or exploiting other vulnerabilities to maintain access to the compromised system.

Key techniques involved in crafting and sending malicious requests to exploit SSRF vulnerabilities include:

- \*\*URL Manipulation\*\*: Modifying the URL or parameters of the server's requests to direct them to desired internal or external resources.

- \*\*Protocol Smuggling\*\*: Exploiting inconsistencies in how the server handles different protocols (e.g., HTTP vs. FTP) to bypass security controls or access restricted resources.

- \*\*Request Method Tampering\*\*: Modifying the HTTP request method (e.g., changing from GET to POST) to bypass restrictions or access additional functionality.

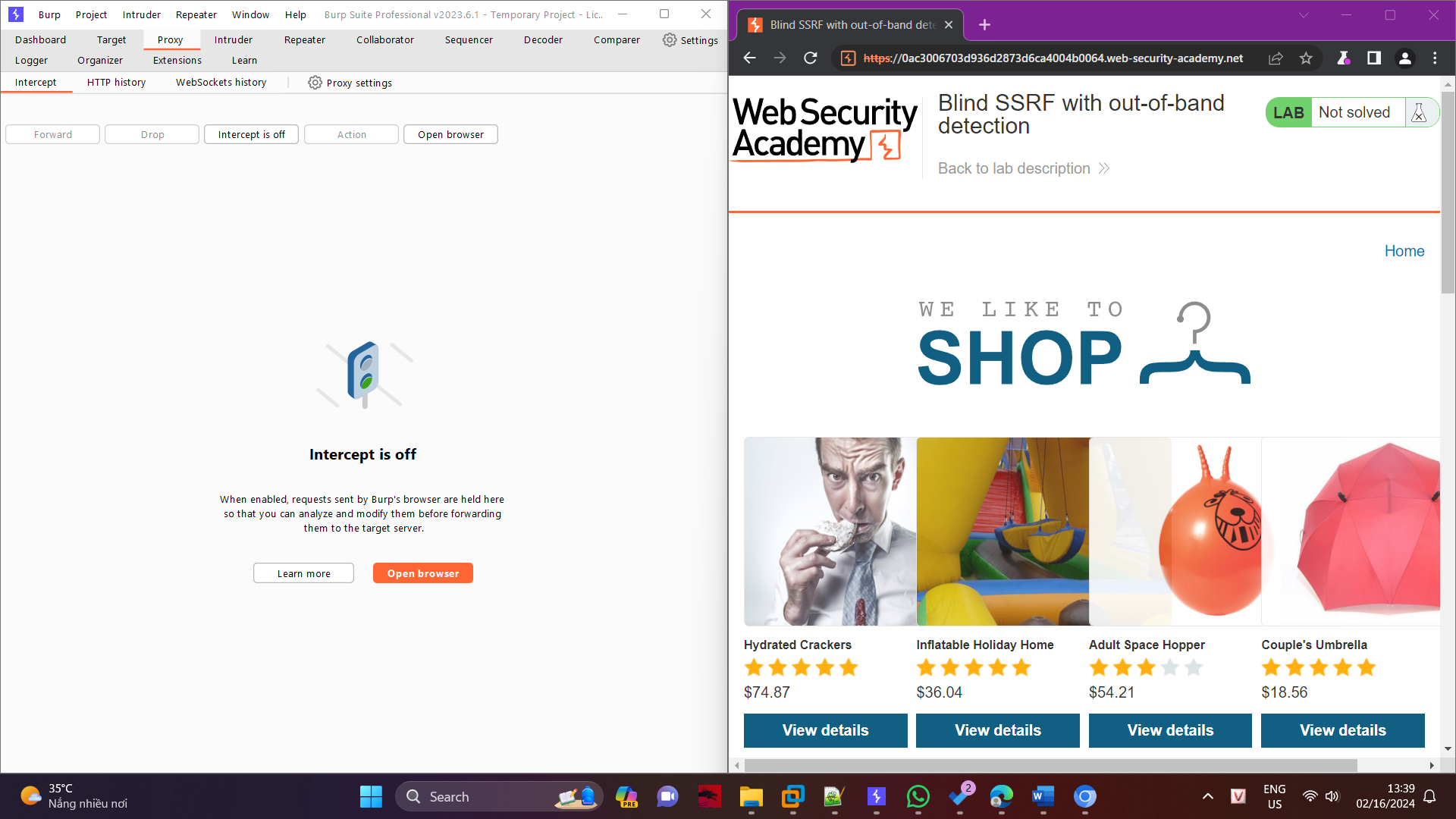
- \*\*IP Address Bypassing\*\*: Using techniques such as DNS rebinding or IP address spoofing to bypass IP-based access controls or restrictions.

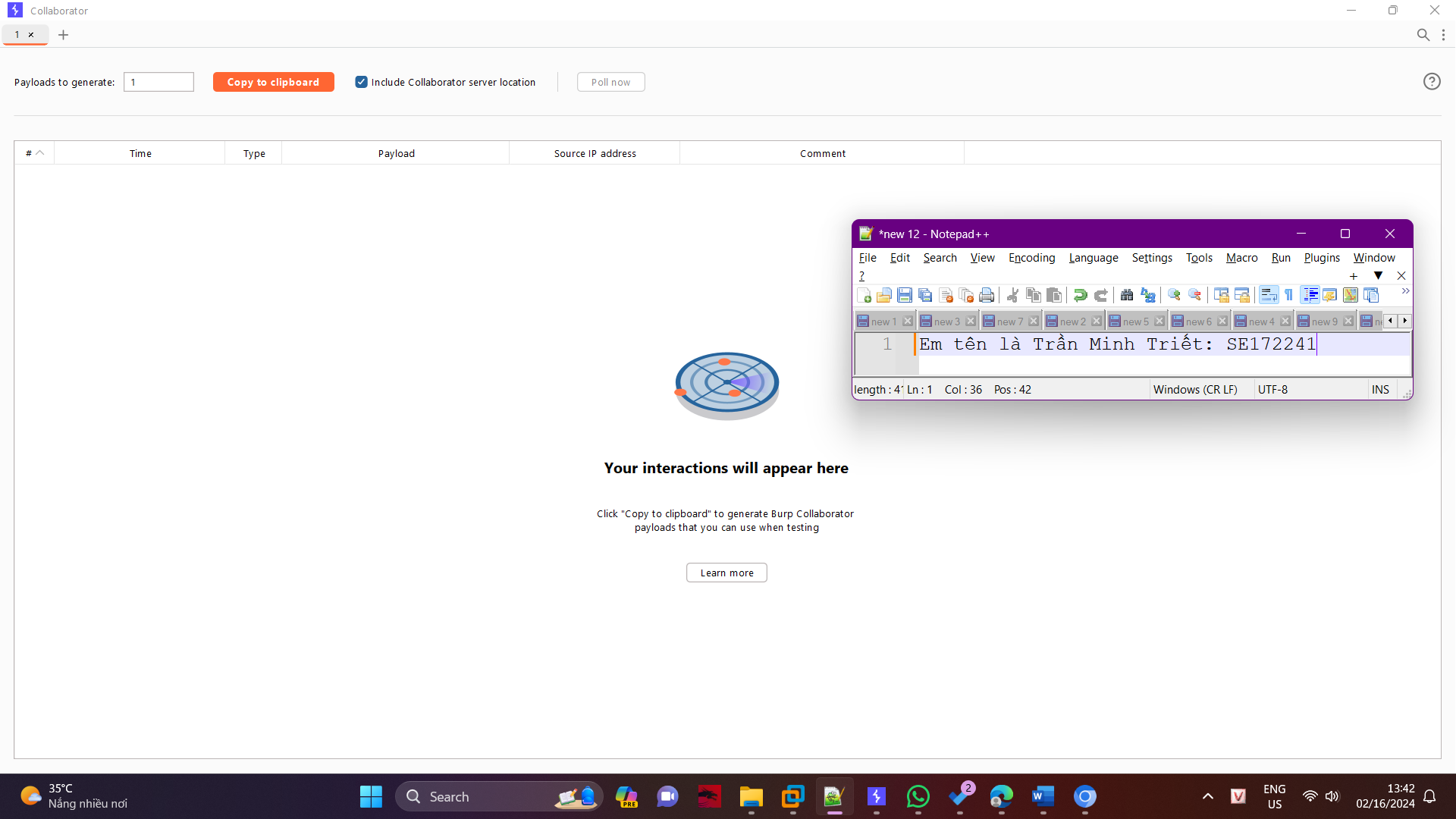
- \*\*Request Body Tampering\*\*: Injecting malicious content into the request body to exploit vulnerabilities in backend services or APIs.

- \*\*Server-Side File Inclusion (SSRFi)\*\*: Exploiting SSRF vulnerabilities to include and execute arbitrary files from the server's file system.

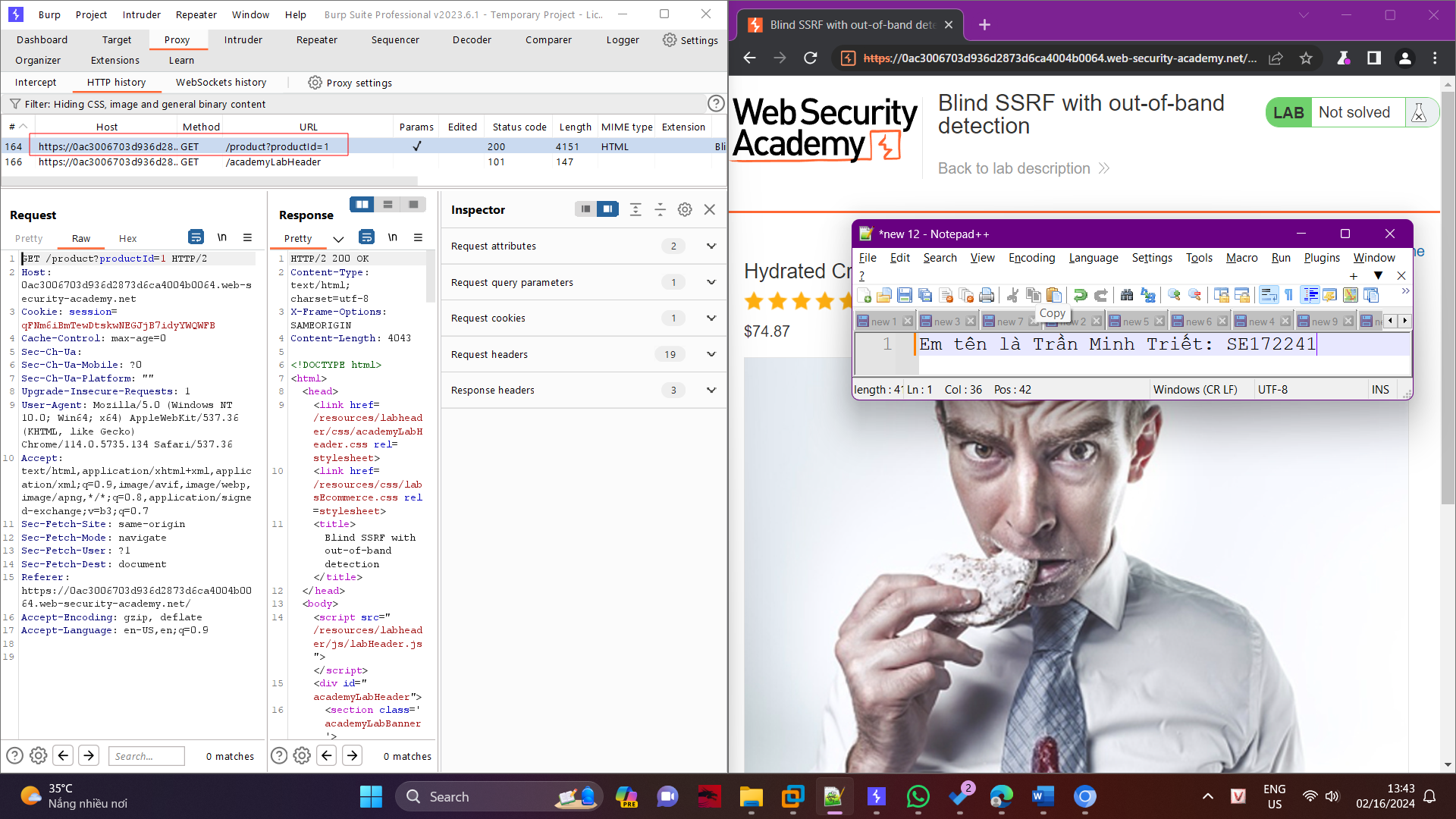
Overall, exploiting an SSRF vulnerability requires a combination of reconnaissance, careful crafting of malicious requests, and understanding of the target environment to effectively manipulate the server into performing unintended actions.

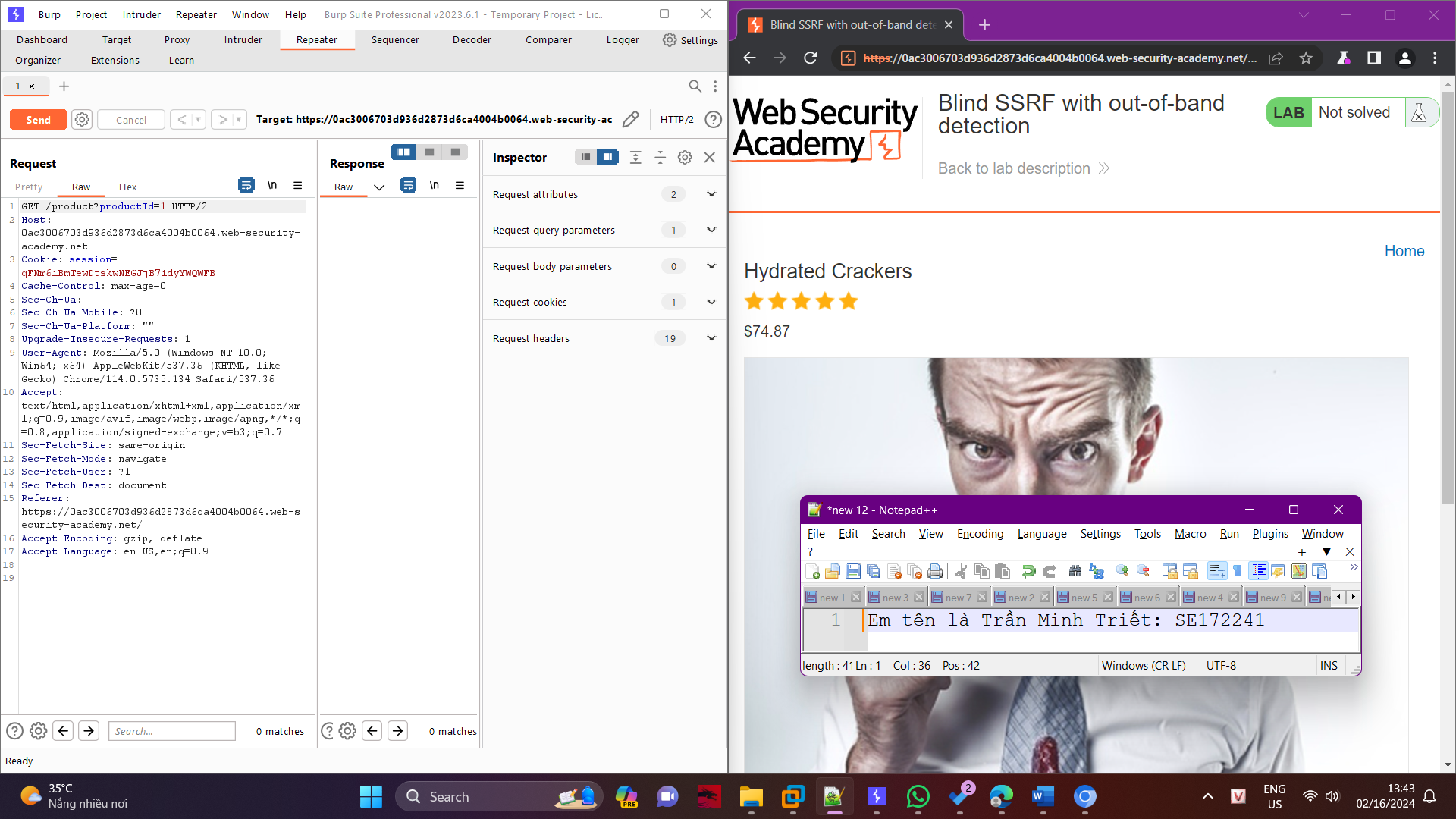
**Challenge**



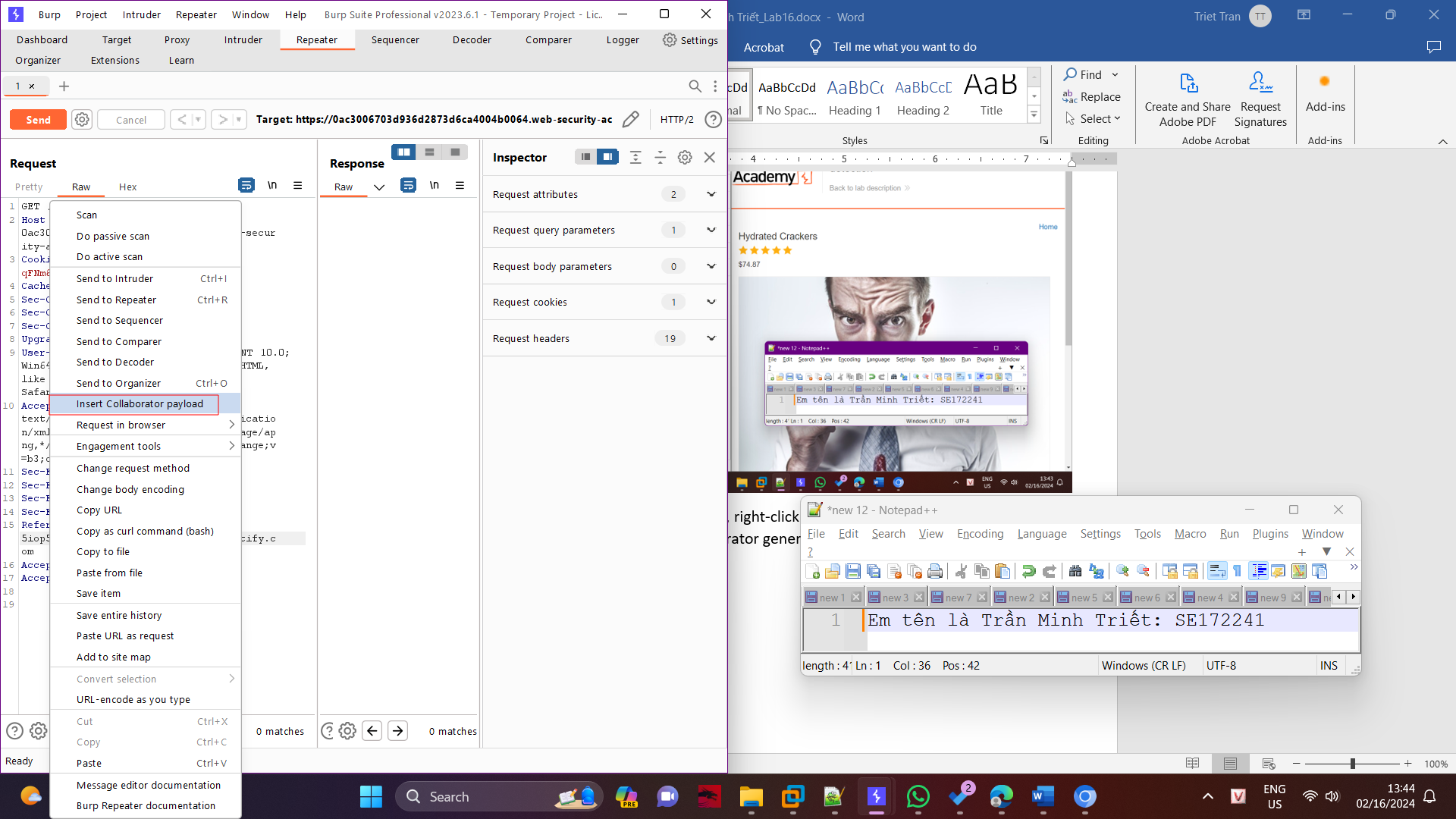


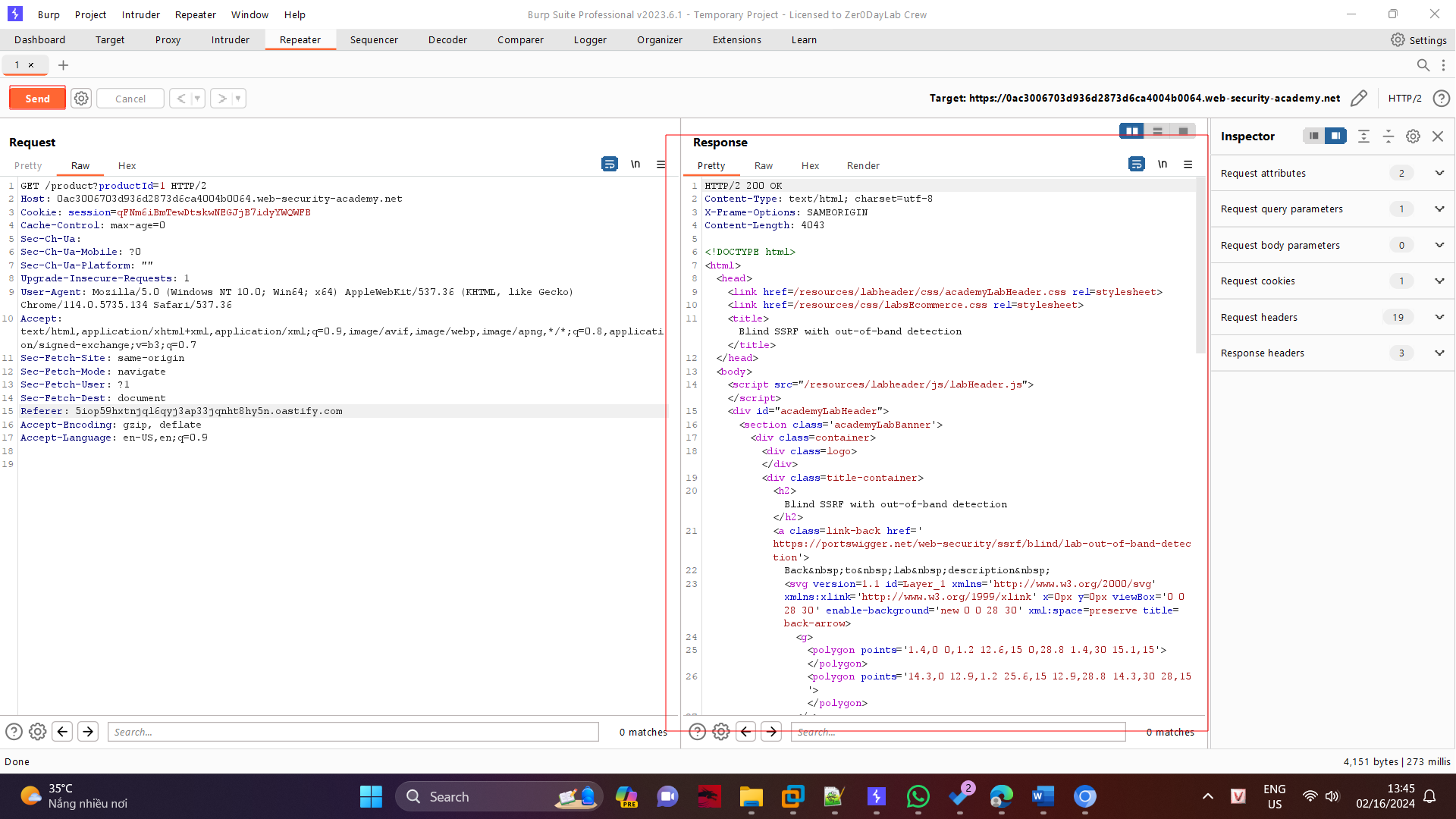
Visit a product, intercept the request in Burp Suite, and send it to Burp Repeater.





Go to the Repeater tab. Select the Referer header, right-click and select "Insert Collaborator Payload" to replace the original domain with a Burp Collaborator generated domain. Send the request.





Go to the Collaborator tab, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again, since the server-side command is executed asynchronously.

