**Lab 17: Race conditions vulnerabilities**

**Objective**:

* This lab is designed to explore Race Conditions vulnerabilities in web applications and systems. Participants will interact with a simulated environment where specific processes or actions are prone to race conditions. The objective is to understand how race conditions occur, identify their impact on application security and data integrity, exploit these vulnerabilities, and learn effective strategies to prevent them.

In this lab, students need to:

* Answer the following questions:
  + What are Race Conditions vulnerabilities, and how do they differ from other types of vulnerabilities in web applications? Explain the concept of a race condition, including how concurrent processes or threads can lead to unexpected or insecure states.
  + Describe the process of exploiting a Race Conditions vulnerability in a web application or system. What techniques can an attacker use to induce a race condition, and how can such vulnerabilities lead to security breaches or data corruption?
* Perform challenge:
  + [Limit overrun race conditions](https://portswigger.net/web-security/race-conditions/lab-race-conditions-limit-overrun)
* 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 17: Race conditions vulnerabilities**

**What are Race Conditions vulnerabilities, and how do they differ from other types of vulnerabilities in web applications? Explain the concept of a race condition, including how concurrent processes or threads can lead to unexpected or insecure states.**

Race Conditions vulnerabilities are a type of security flaw that occurs in software when the outcome of a program depends on the sequence or timing of events, particularly in a multi-threaded or concurrent environment. In web applications, race conditions can lead to unexpected or insecure states by allowing multiple processes or threads to access and modify shared resources simultaneously, leading to inconsistent or unpredictable behavior.

The concept of a race condition arises from the concurrent execution of multiple processes or threads, where the outcome of a program depends on the timing or order in which these processes or threads are executed. A race condition occurs when two or more processes or threads attempt to access and modify a shared resource concurrently, resulting in unexpected or insecure behavior.

Here's how a race condition typically occurs:

1. \*\*Shared Resource\*\*: The software contains a shared resource, such as a file, database record, or variable, that multiple processes or threads need to access and modify.

2. \*\*Non-Atomic Operations\*\*: The operations performed on the shared resource are not atomic, meaning they cannot be guaranteed to execute as a single, indivisible unit. Instead, they consist of multiple steps that can be interrupted or interleaved with operations from other processes or threads.

3. \*\*Concurrency\*\*: Multiple processes or threads attempt to access and modify the shared resource simultaneously, without proper synchronization or coordination.

4. \*\*Unpredictable Outcome\*\*: Depending on the timing and order of operations, the outcome of the program can vary. In some cases, conflicting modifications to the shared resource may lead to data corruption, inconsistency, or unexpected behavior.

Race conditions differ from other types of vulnerabilities in web applications in several ways:

- \*\*Concurrency\*\*: Race conditions specifically arise from the concurrent execution of multiple processes or threads, whereas other vulnerabilities may not depend on concurrent execution.

- \*\*Timing Dependencies\*\*: Race conditions depend on the timing or order of events, where the outcome of a program can vary depending on how processes or threads are scheduled to execute.

- \*\*Shared Resources\*\*: Race conditions involve shared resources that are accessed and modified by multiple processes or threads simultaneously, whereas other vulnerabilities may involve different types of security flaws, such as input validation errors, insecure configurations, or logic flaws.

- \*\*Unpredictable Behavior\*\*: Race conditions can lead to unpredictable or inconsistent behavior, making them challenging to detect and mitigate compared to other vulnerabilities with more deterministic effects.

Overall, race conditions pose a significant risk to the security and reliability of software systems, including web applications, and require careful design, implementation, and testing to prevent and mitigate. Strategies for addressing race conditions include proper synchronization, mutual exclusion, and access control mechanisms to ensure safe access to shared resources in multi-threaded or concurrent environments.

**Describe the process of exploiting a Race Conditions vulnerability in a web application or system. What techniques can an attacker use to induce a race condition, and how can such vulnerabilities lead to security breaches or data corruption?**

Exploiting a Race Conditions vulnerability in a web application or system involves manipulating the timing and order of operations to achieve unintended or insecure behavior. Here's a general process an attacker might follow to exploit a race condition:

1. \*\*Identify Race Condition\*\*: The attacker first needs to identify a vulnerable portion of code within the web application or system where a race condition exists. This could involve analyzing the application's source code, observing its behavior under load, or conducting penetration testing to identify potential concurrency issues.

2. \*\*Determine Targeted Resources\*\*: The attacker identifies the shared resources that are accessed and modified by multiple processes or threads concurrently. These resources could include files, database records, variables, or any other data structures that are not properly synchronized or protected against simultaneous access.

3. \*\*Craft Attack Scenarios\*\*: The attacker devises attack scenarios that exploit the race condition to achieve their objectives. This could involve scenarios where conflicting operations on the shared resource lead to data corruption, privilege escalation, unauthorized access, or other security breaches.

4. \*\*Induce Race Condition\*\*: The attacker leverages techniques to induce the race condition by manipulating the timing and order of operations. This could involve sending multiple simultaneous requests, injecting delays or timing attacks, or exploiting concurrency-related vulnerabilities in the application's logic or design.

5. \*\*Exploit Vulnerabilities\*\*: Once the race condition is induced, the attacker exploits the vulnerable state to achieve their objectives. This could involve various actions depending on the nature of the vulnerability and the attacker's goals, such as modifying critical data, bypassing access controls, escalating privileges, or launching further attacks against the system or other users.

Techniques for inducing a race condition and exploiting vulnerabilities include:

- \*\*Concurrency Attacks\*\*: Sending multiple simultaneous requests or performing actions in parallel to increase the likelihood of race condition occurrences.

- \*\*Delay Injection\*\*: Injecting delays or timing attacks to manipulate the timing and order of operations and increase the likelihood of race conditions.

- \*\*Time-of-Check to Time-of-Use (TOCTOU) Attacks\*\*: Exploiting vulnerabilities where the state of a resource changes between the time it is checked and the time it is used, leading to unexpected or insecure behavior.

- \*\*Cache Poisoning\*\*: Exploiting caching mechanisms to manipulate the state of shared resources and induce race conditions.

- \*\*Resource Exhaustion\*\*: Overloading the system with concurrent requests or resource-intensive operations to increase the likelihood of race conditions and exploit vulnerabilities.

Once exploited, race conditions can lead to security breaches or data corruption by allowing attackers to:

- Modify critical data or configurations in an unpredictable or unauthorized manner.

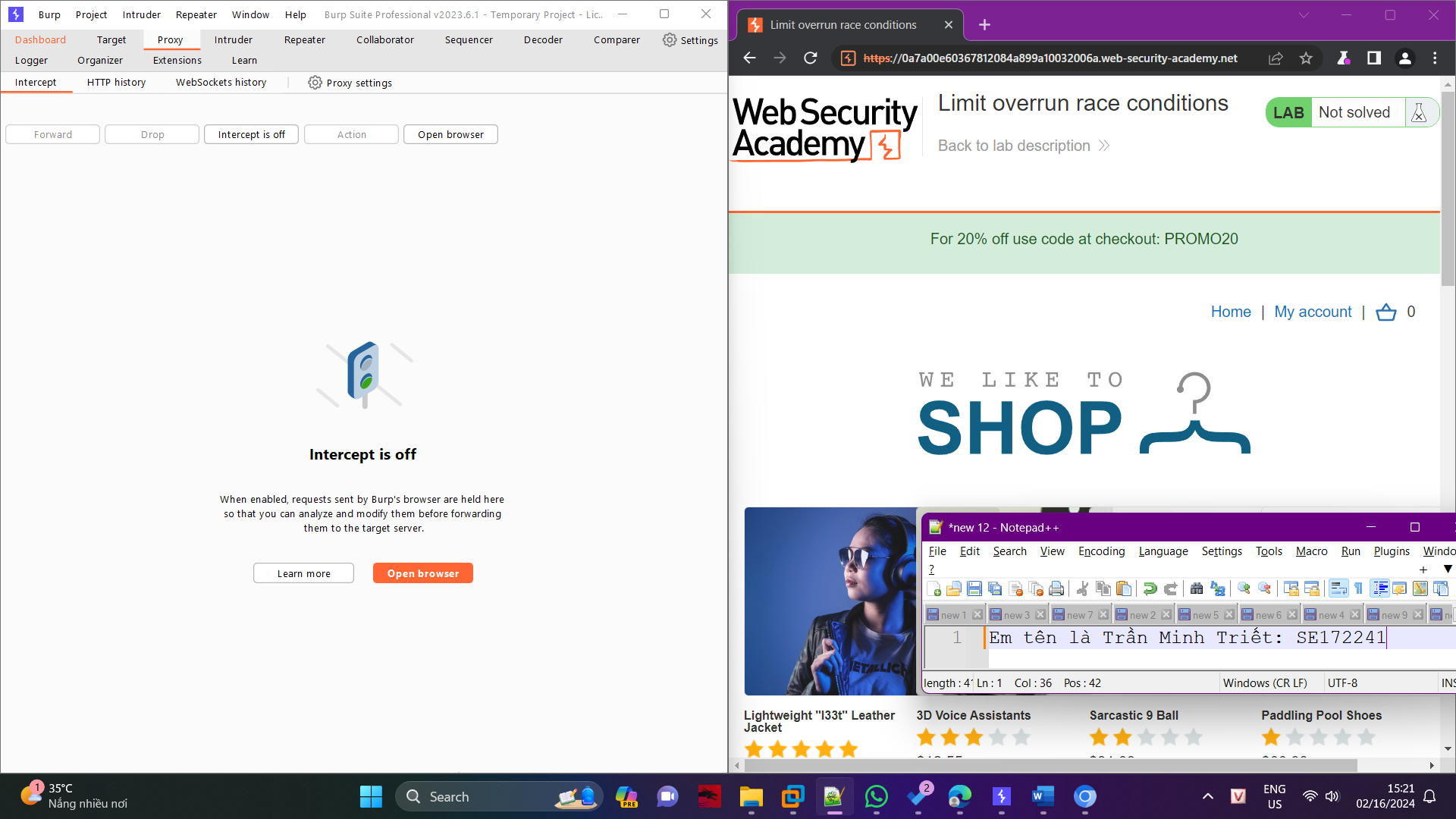
- Bypass access controls or privilege escalation to gain unauthorized access to sensitive resources or functionality.

- Cause denial-of-service conditions or system instability by corrupting data or disrupting normal operations.

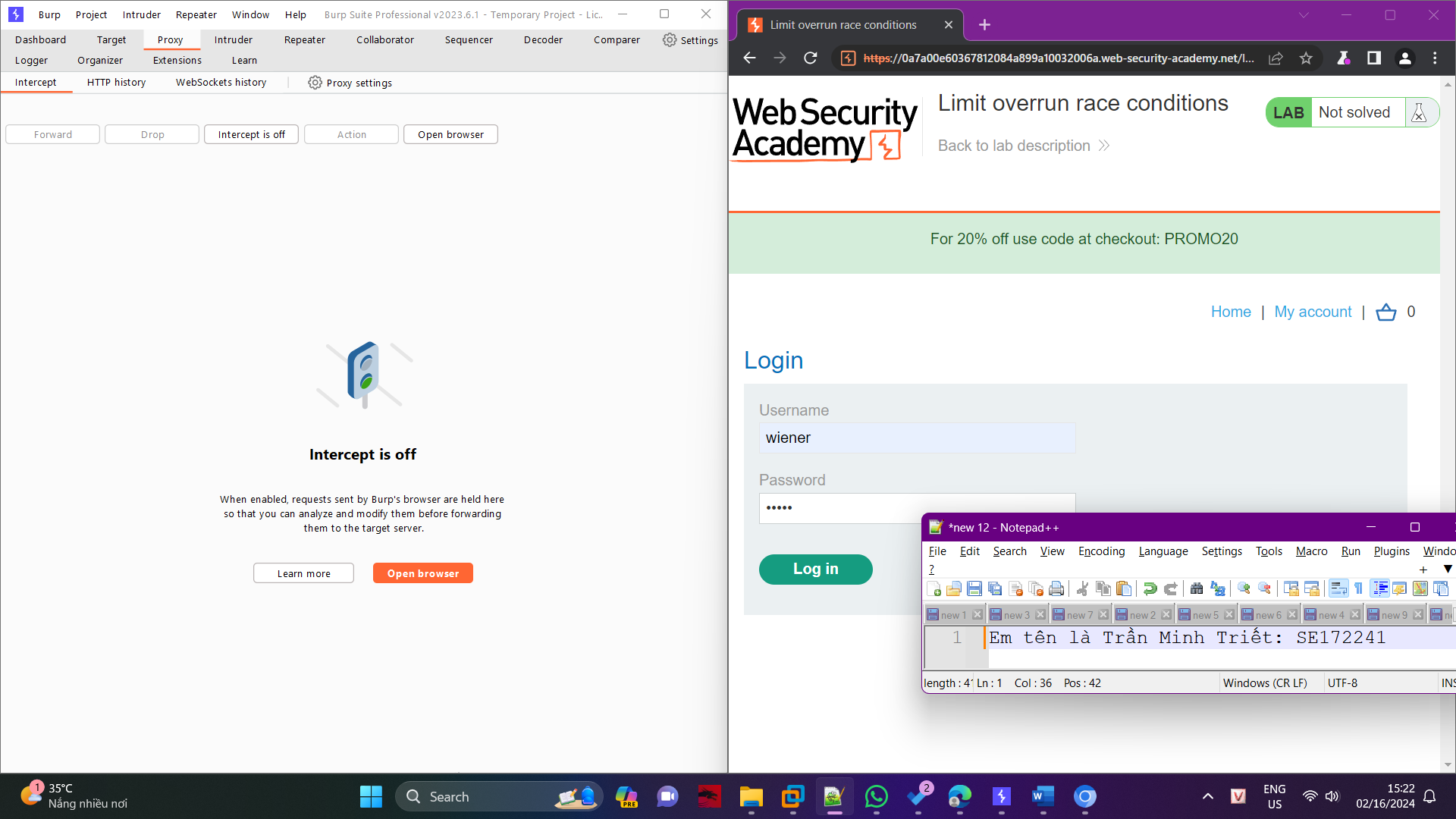
- Exfiltrate sensitive information or manipulate the behavior of the application or system to achieve malicious objectives.

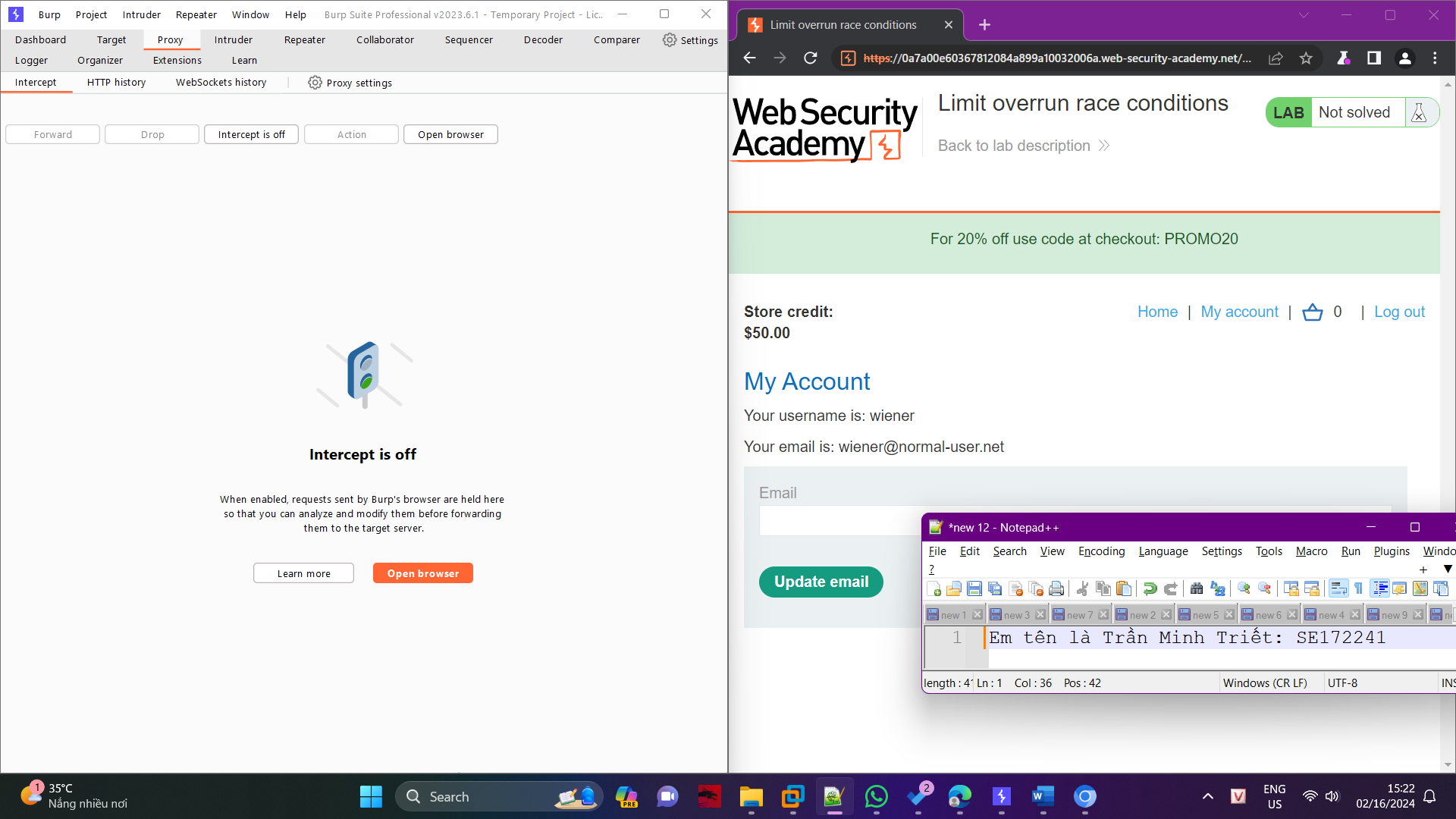
Overall, mitigating race conditions requires careful design, implementation, and testing of concurrency control mechanisms to ensure the safe and consistent access to shared resources in multi-threaded or concurrent environments.

**Challenge**

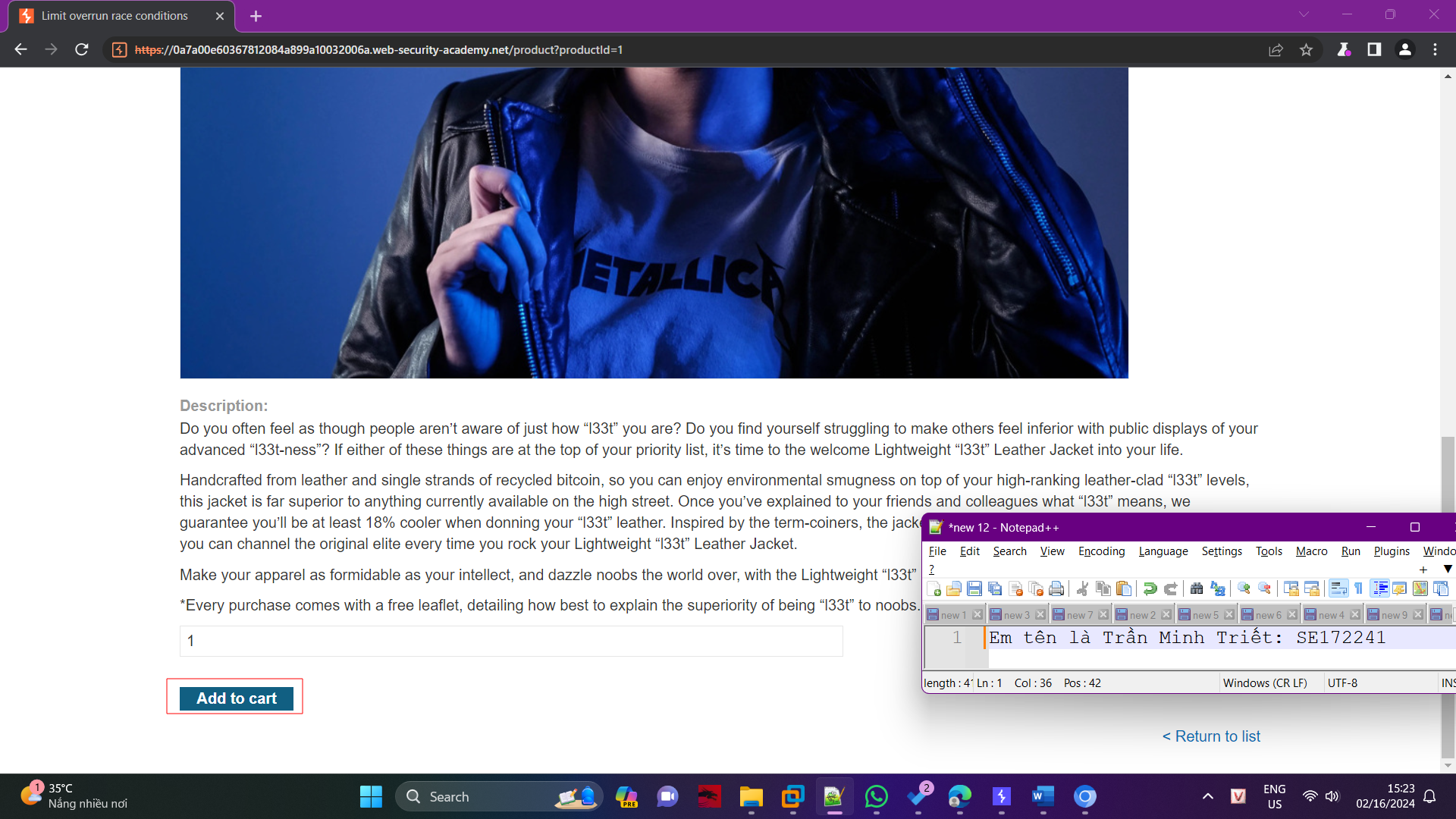


Log In to your account using the credentials wiener:peter

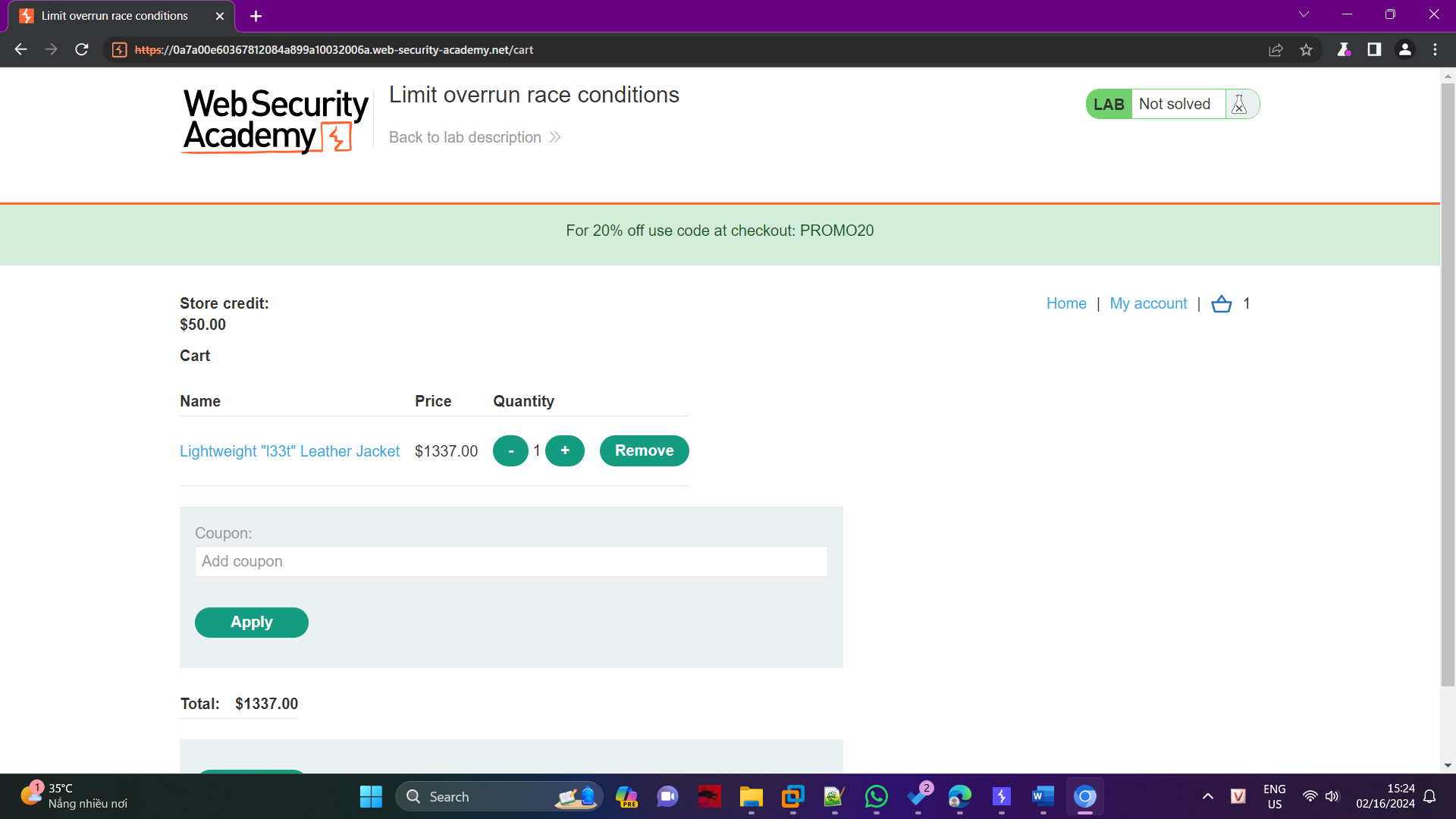


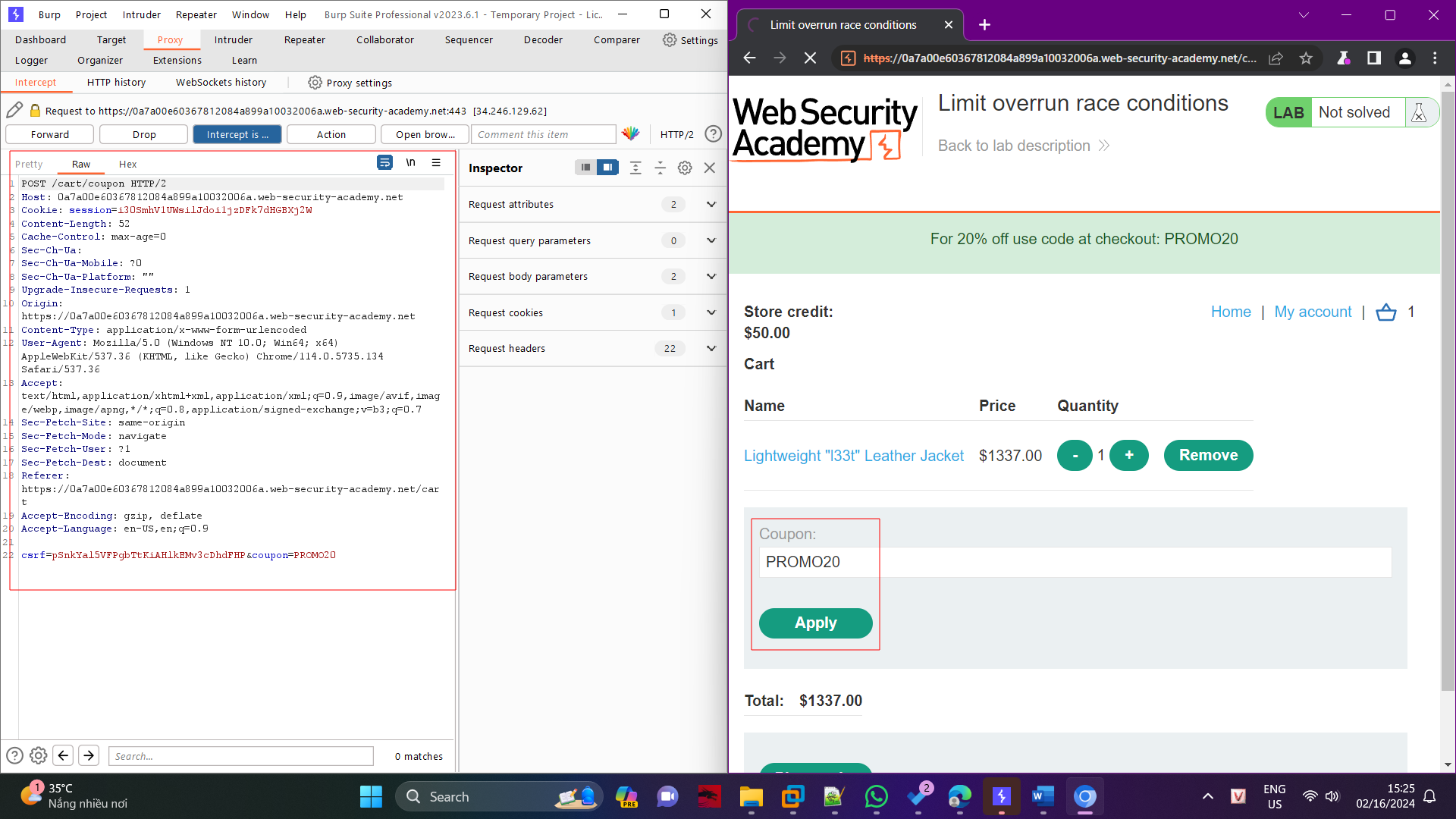


Add Lightweight L33t Leather Jacket to the cart

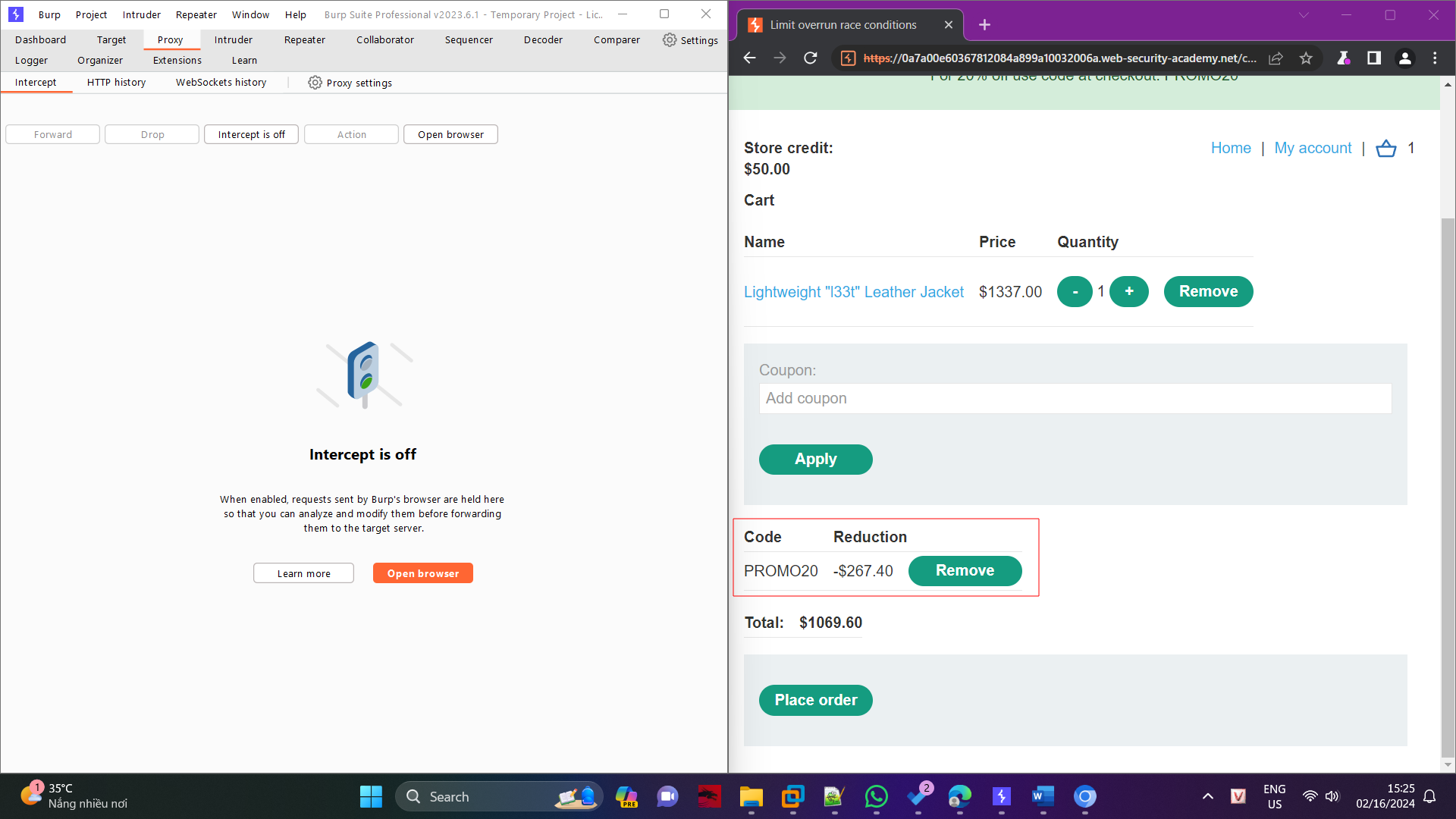


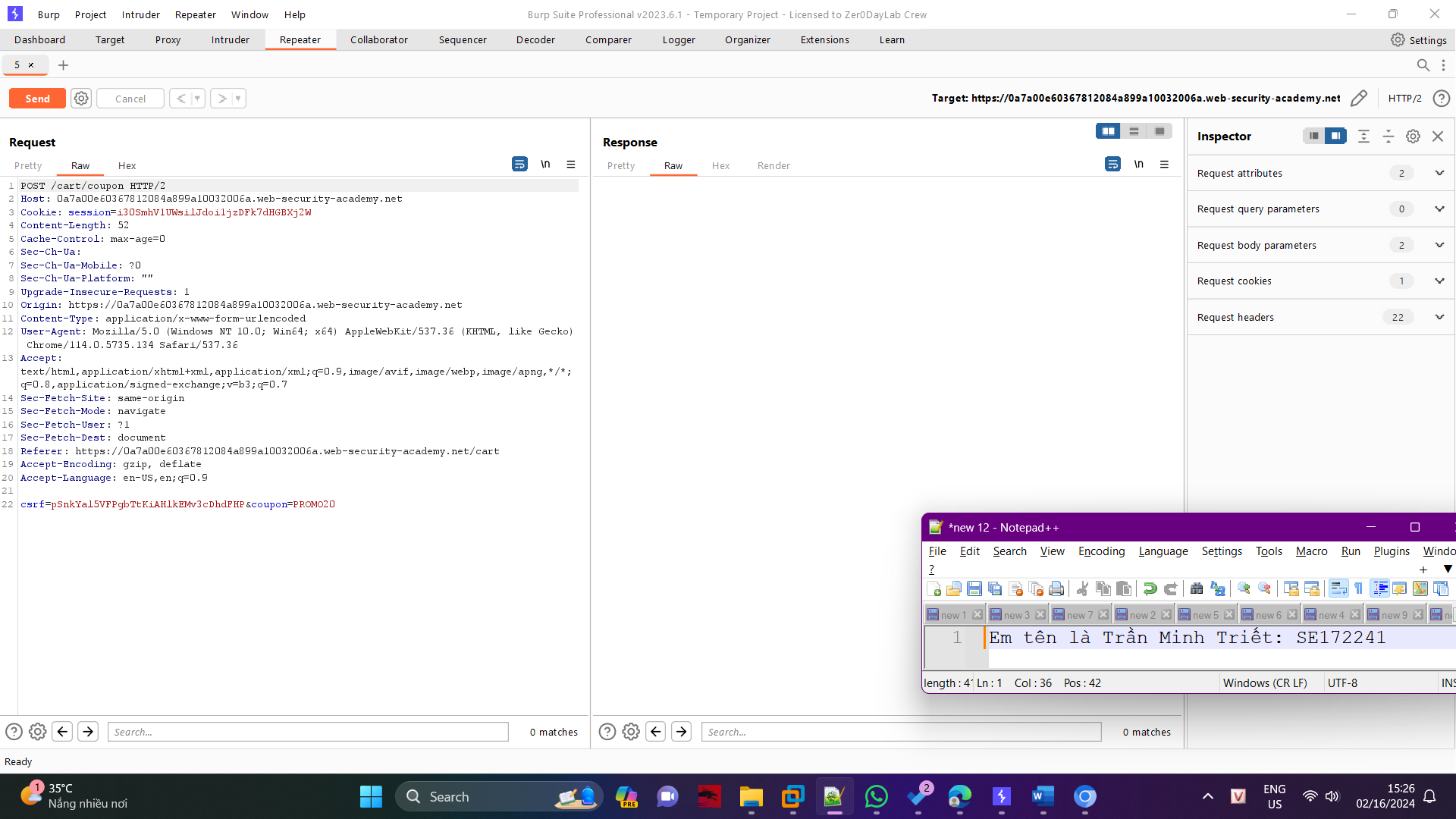
Now Apply the coupon and capture the request on Burp



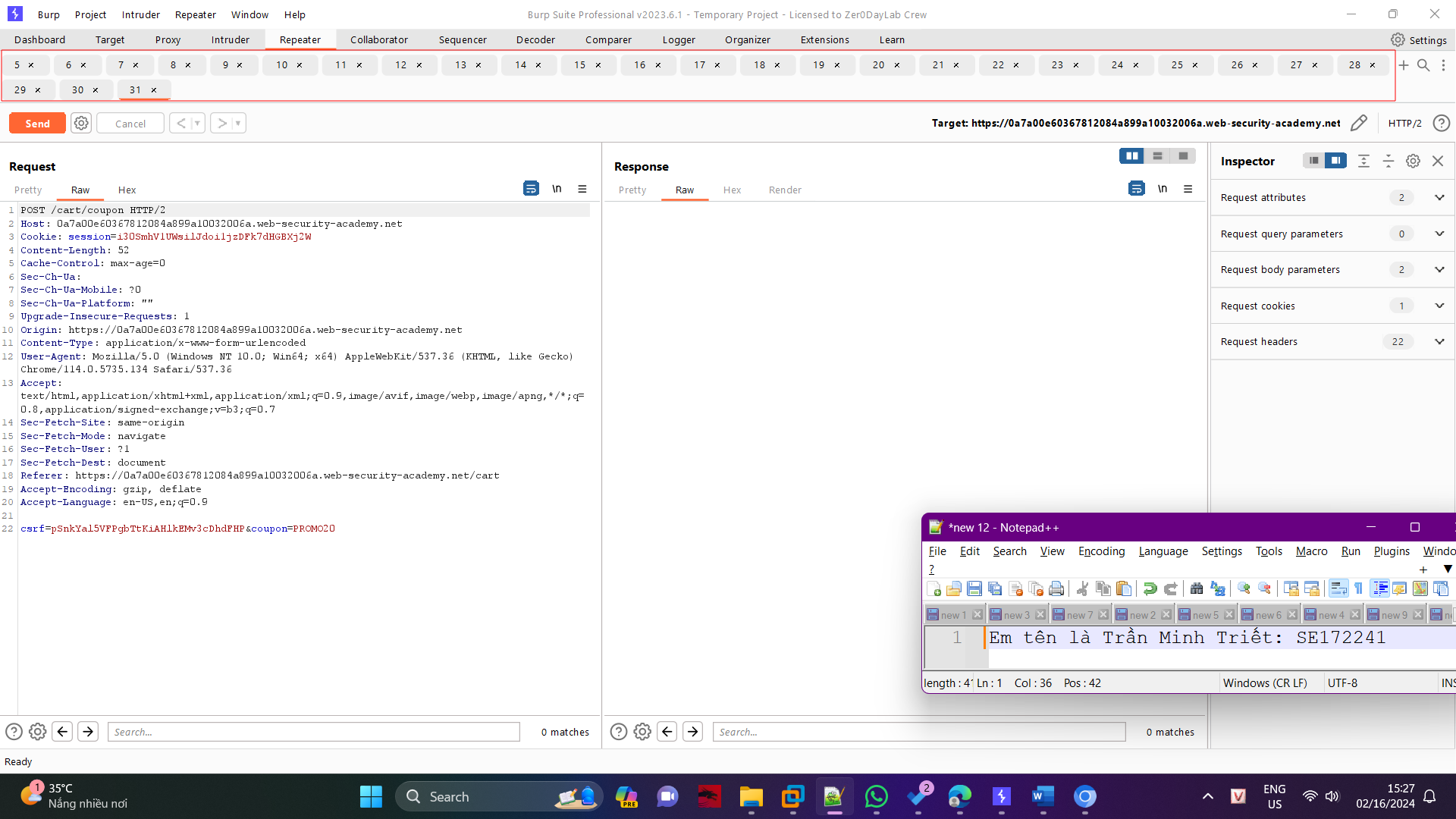


Now send it to the repeater and turn off the Intercept

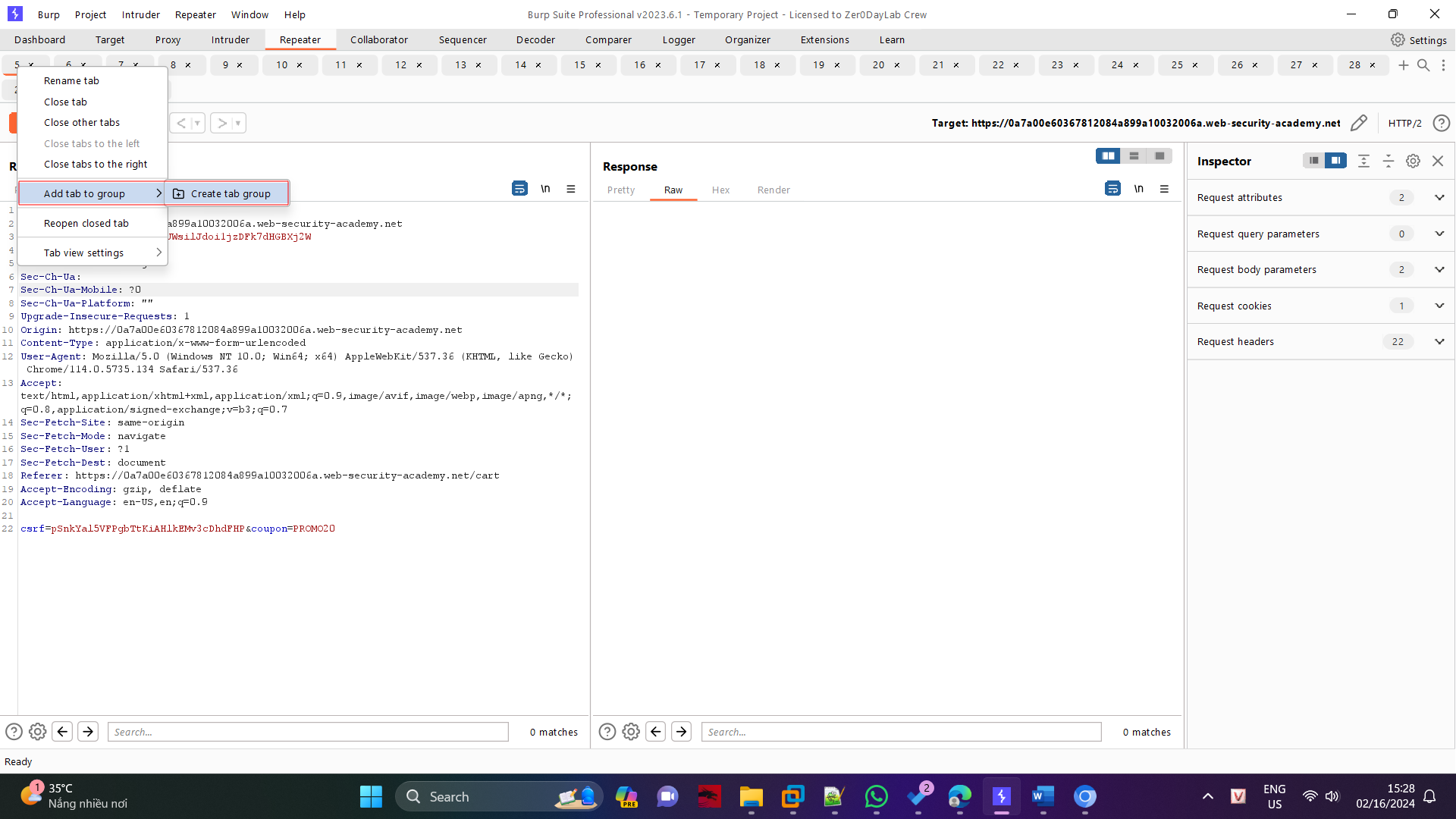


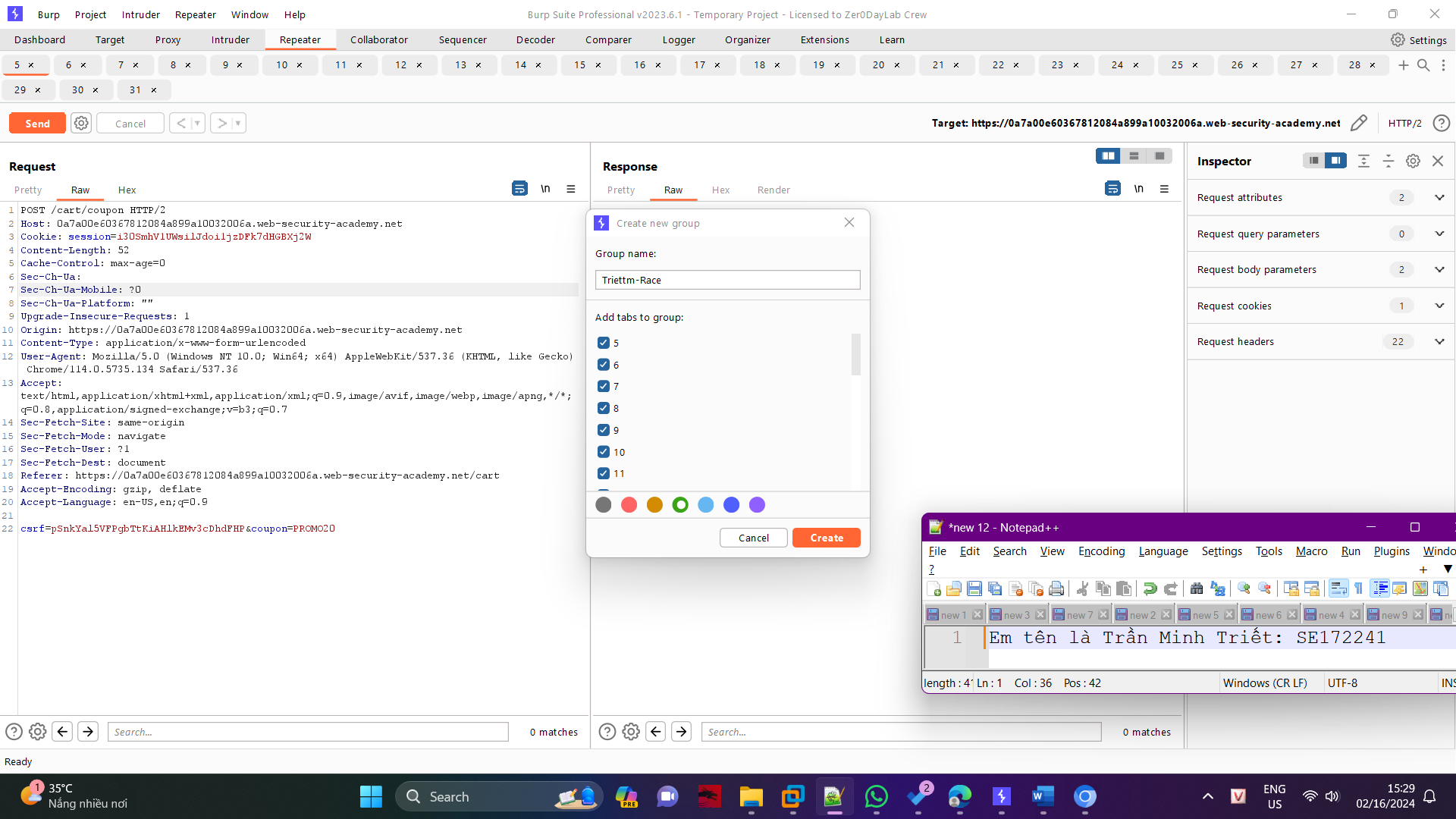


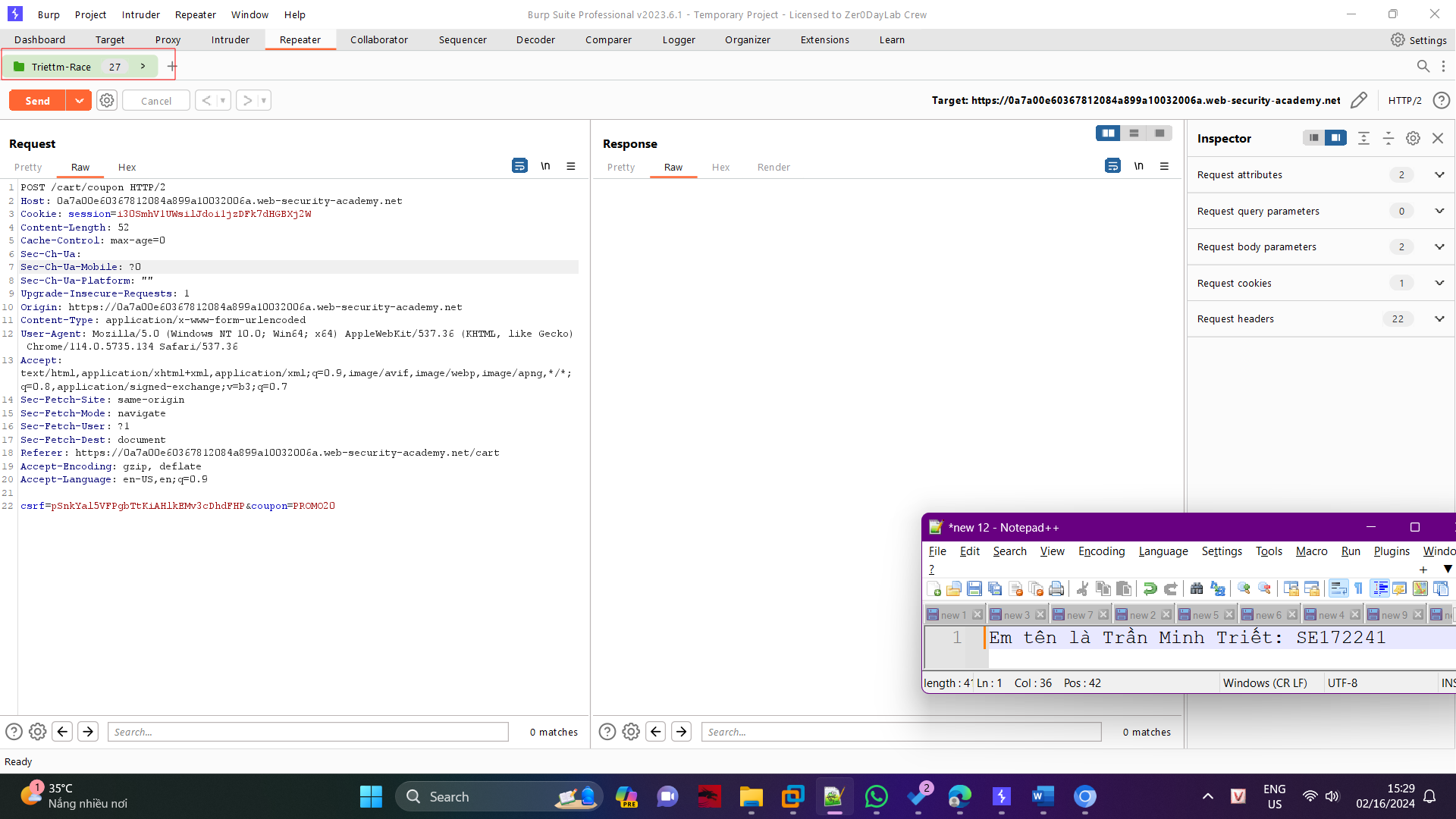
Now click on the request and send it again to the repeater itself 27 times totally



Now Right Click on any tab then click > add tab to group > create tab group







Now, go to the browser and remove any existing coupon code

On the Burp repeater, right-click send and click send group in parallel

