**Họ và tên: Trần Minh Triết**

**MSSV: SE172241**

**Class: IA1702**

**Question:**

**What is OAuth, and how does it facilitate authorization in modern web applications?**

OAuth (Open Authorization) is an open standard framework that allows users to grant third-party applications limited access to their resources without revealing their credentials (such as passwords). It is commonly used as a way to facilitate secure authorization in modern web applications and APIs.

OAuth operates by defining a set of roles, protocols, and endpoints to enable the authorization process. The main actors in the OAuth process are:

Resource Owner (User): The entity that owns the protected resources (e.g., a user with an account on a social media platform).

Client: The application or service that is requesting access to the user's resources. This could be a mobile app, a web application, or any other type of software.

Authorization Server: The server that authenticates the resource owner, obtains their consent, and issues access tokens to the client.

Resource Server: The server that hosts the protected resources. It is separate from the authorization server, although they can be the same entity in some cases.

The OAuth process typically involves the following steps:

Authorization Request: The client initiates the process by redirecting the user to the authorization server with a request for access. This request includes the client's identity, scope of access requested, and a redirect URI where the user will be sent after authentication.

User Authentication and Consent: The authorization server authenticates the user and, if successful, presents them with a consent screen specifying the requested permissions. The user can then approve or deny the request.

Authorization Grant: If the user grants permission, the authorization server issues an authorization grant to the client. The grant is a token that represents the user's consent.

Token Request: The client presents the authorization grant to the authorization server and requests an access token.

Token Response: If the authorization server validates the request, it issues an access token to the client. This token is used by the client to access the protected resources on behalf of the user.

Accessing Protected Resources: The client uses the access token to make requests to the resource server, which validates the token and provides access to the requested resources.

OAuth provides a secure and standardized way for users to grant limited access to their resources without sharing sensitive information. It is widely used in the context of social media logins, third-party API integrations, and other scenarios where secure access to user data is required.

**Describe how consent phishing can be executed in an OAuth context. How can attackers leverage OAuth permissions to gain unauthorized access to user data? Discuss strategies that can be employed to detect and prevent consent phishing in OAuth implementations.**

Consent phishing in an OAuth context involves tricking users into granting unauthorized access to their accounts and data by manipulating the OAuth authorization process. Attackers use social engineering techniques to deceive users into providing their consent, leading to the compromise of sensitive information. Here's how consent phishing can be executed and some strategies to detect and prevent it:

Execution of Consent Phishing:

Fake Authorization Screens: Attackers may create fake authorization screens that mimic legitimate OAuth consent screens. Users, thinking they are granting access to a legitimate application, may unknowingly provide their credentials and authorize the attacker's application.

Misleading Redirects: Phishers might redirect users to malicious websites that closely resemble the legitimate authorization server. Users may then enter their credentials and consent information, unknowingly granting access to the attacker.

Compromised Third-Party Apps: Attackers can compromise existing, seemingly legitimate third-party applications. Users, trusting the app, might grant permissions, not realizing that they are providing access to an attacker.

Strategies to Detect and Prevent Consent Phishing:

User Education and Awareness:

Educate users about the risks of granting permissions to unfamiliar or suspicious applications.

Encourage users to verify the authenticity of authorization screens by checking the domain and ensuring it is using HTTPS.

Clear and Transparent Authorization Screens:

Authorization screens should clearly state the permissions being requested and explain why the app needs those permissions.

Include visual indicators, such as logos and branding, to help users verify the legitimacy of the authorization screen.

Multi-Factor Authentication (MFA):

Implement multi-factor authentication to add an additional layer of security, even if attackers manage to gain access to the user's credentials.

Monitoring and Anomaly Detection:

Implement monitoring systems to detect unusual patterns of user activity, such as a sudden surge in authorization requests.

Flag and investigate instances where users grant permissions to a new application but rarely or never use it.

User Consent Revocation:

Allow users to easily revoke previously granted permissions through a centralized account management interface.

Regularly remind users to review and audit the applications that have access to their accounts.

Use OAuth 2.0 Security Best Practices:

Follow OAuth 2.0 security best practices, including using secure HTTPS connections, validating redirect URIs, and implementing token expiration and refresh mechanisms.

OAuth Token Scopes:

Be granular with the OAuth token scopes requested. Only request the minimum permissions necessary for the application to function.

Security Reviews of Third-Party Apps:

Conduct regular security reviews of third-party applications that integrate with the OAuth system.

Implement a vetting process for third-party apps before they are allowed to request user permissions.

Continuous Security Training:

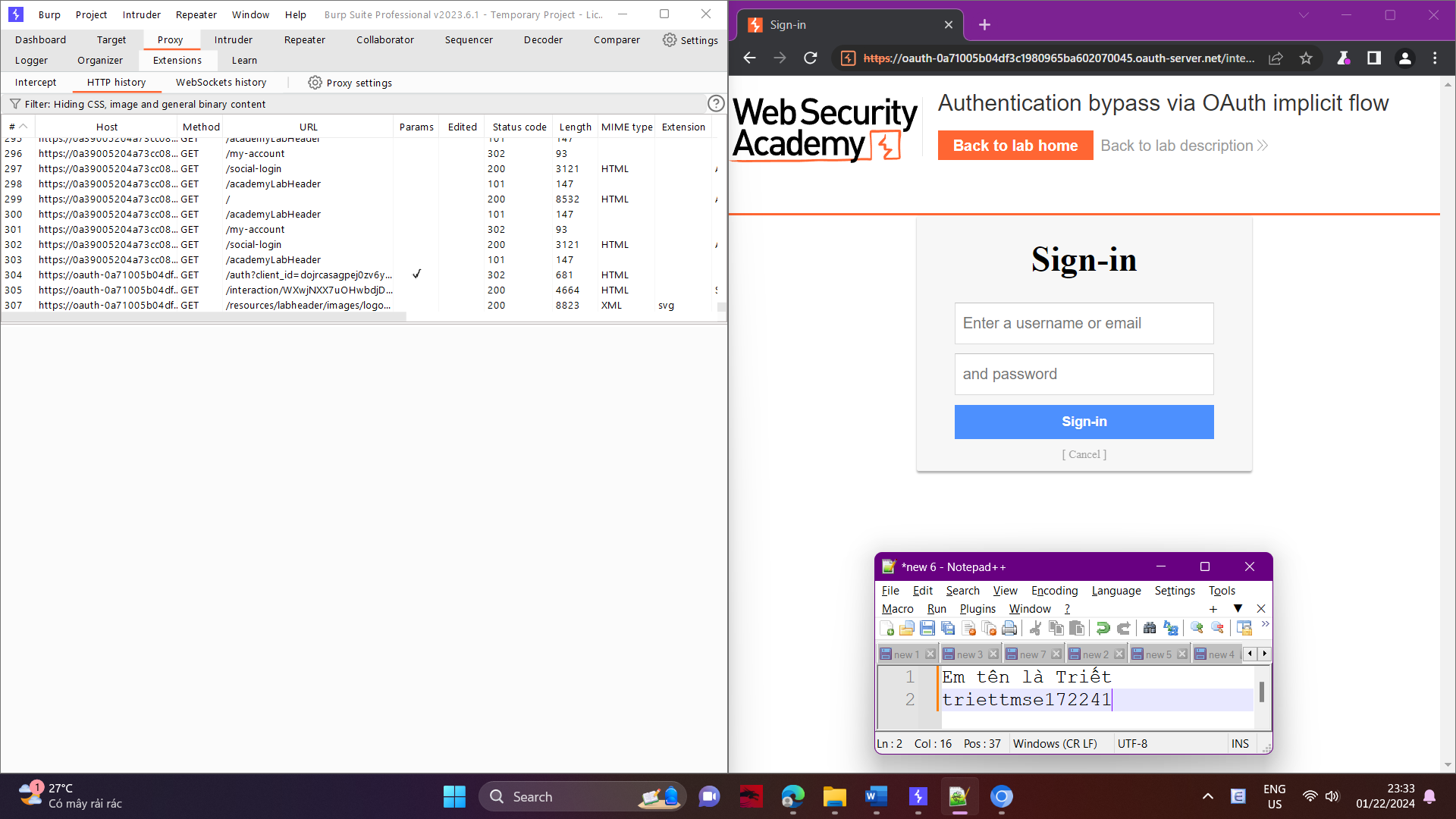
Keep developers and administrators informed about emerging security threats and social engineering tactics, emphasizing the importance of maintaining a secure OAuth implementation.

By combining these strategies, organizations can strengthen their OAuth implementations and reduce the risk of consent phishing attacks. Continuous monitoring, user education, and proactive security measures are essential components of an effective defense against OAuth-related threats.

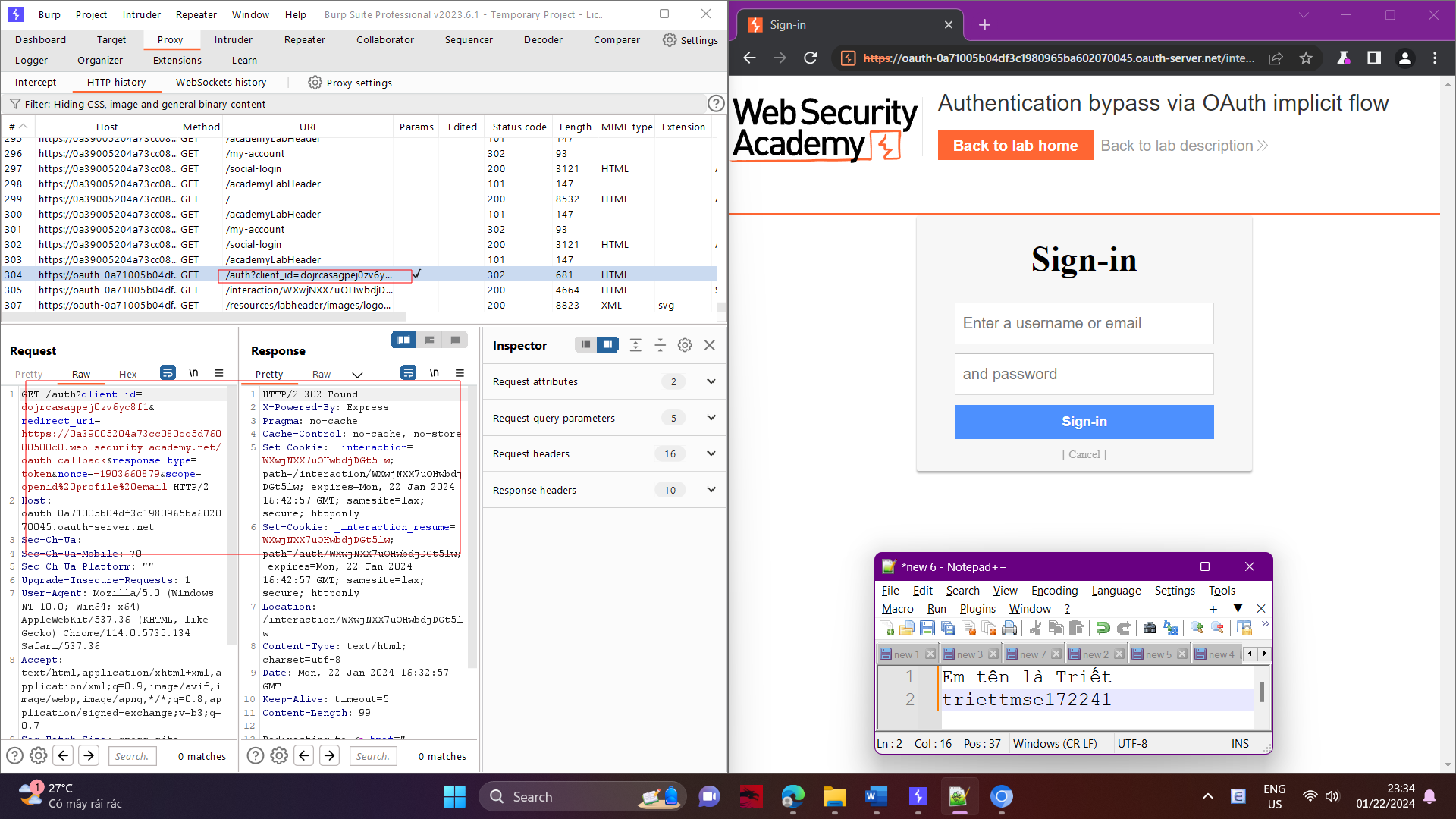
# Lab 8: Access control vulnerabilities - OAuth2.0

# 

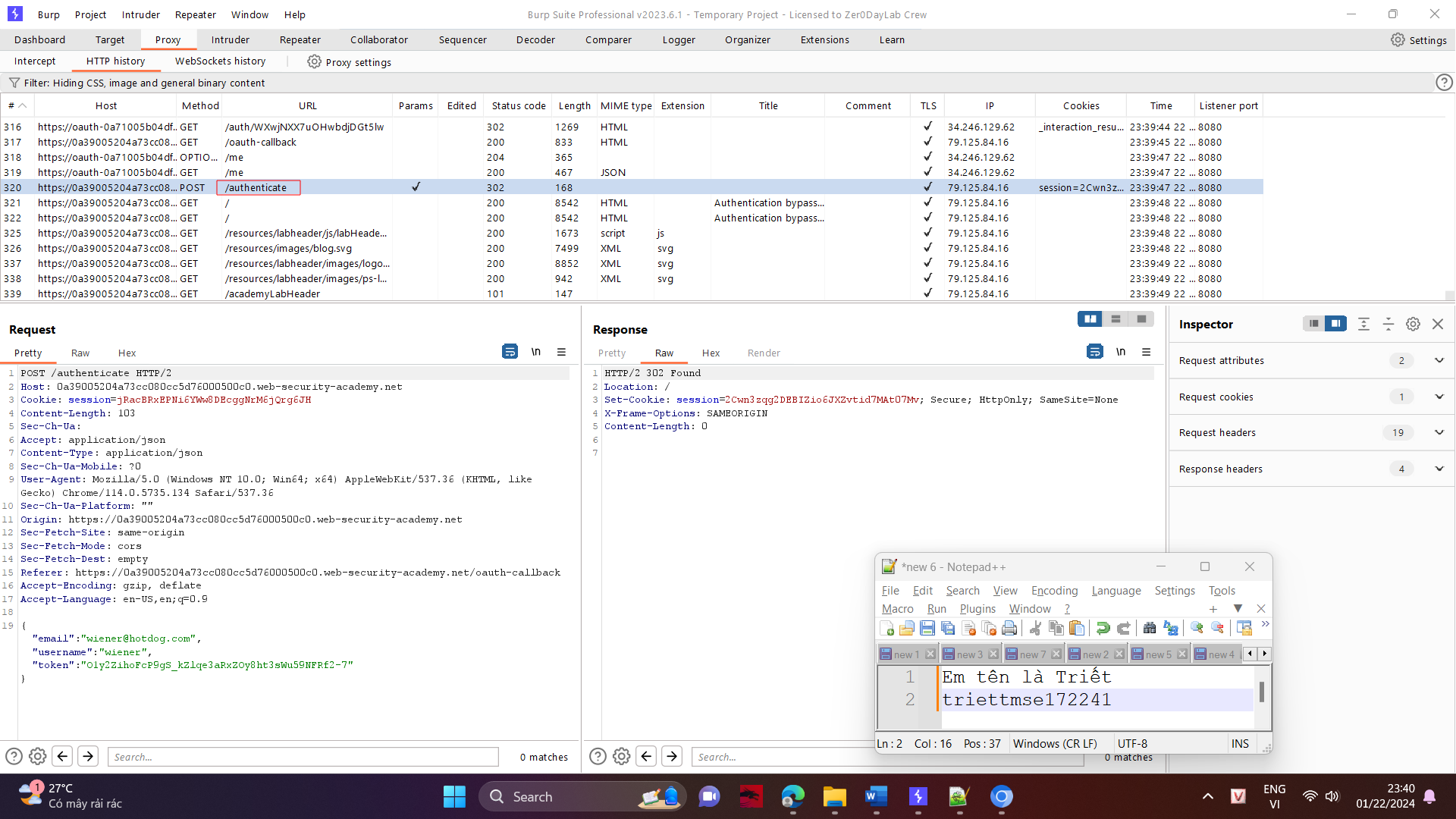
1. While proxying traffic through Burp, click "My account" and complete the OAuth login process. Afterwards, you will be redirected back to the blog website.



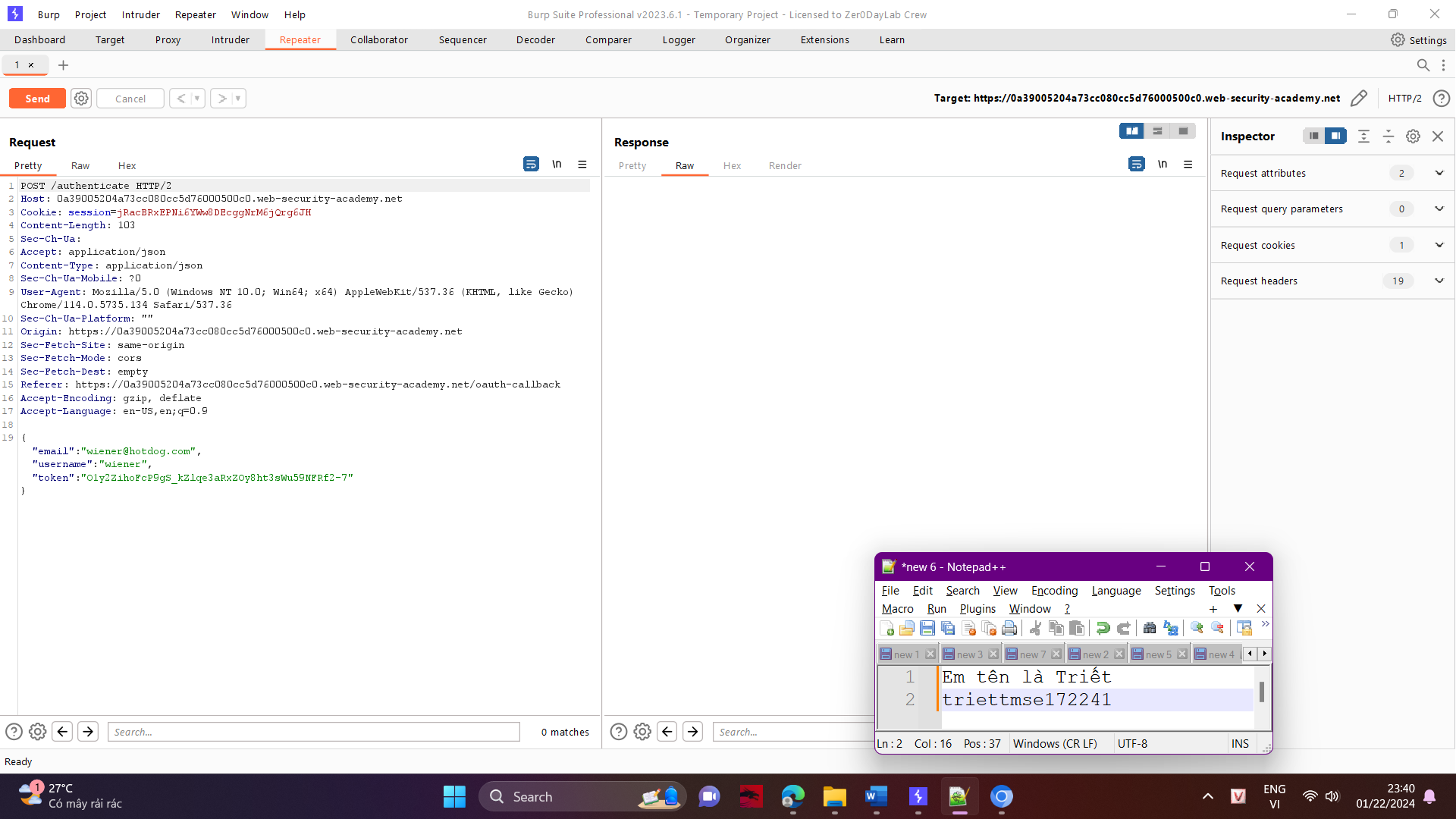
1. In Burp, go to "Proxy" > "HTTP history" and study the requests and responses that make up the OAuth flow. This starts from the authorization request GET /auth?client\_id=[...].



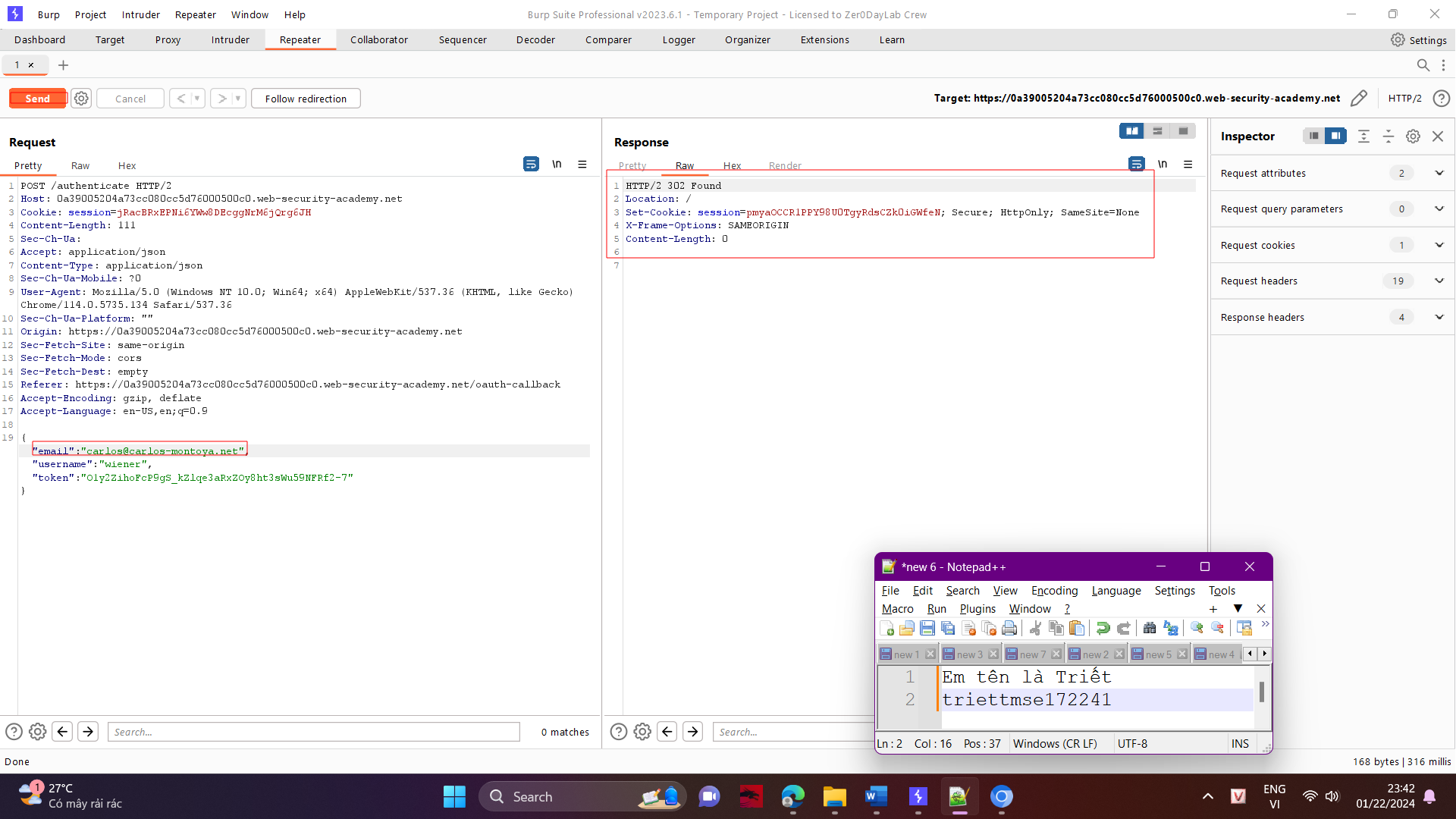
1. Notice that the client application (the blog website) receives some basic information about the user from the OAuth service. It then logs the user in by sending a POST request containing this information to its own /authenticate endpoint, along with the access token.

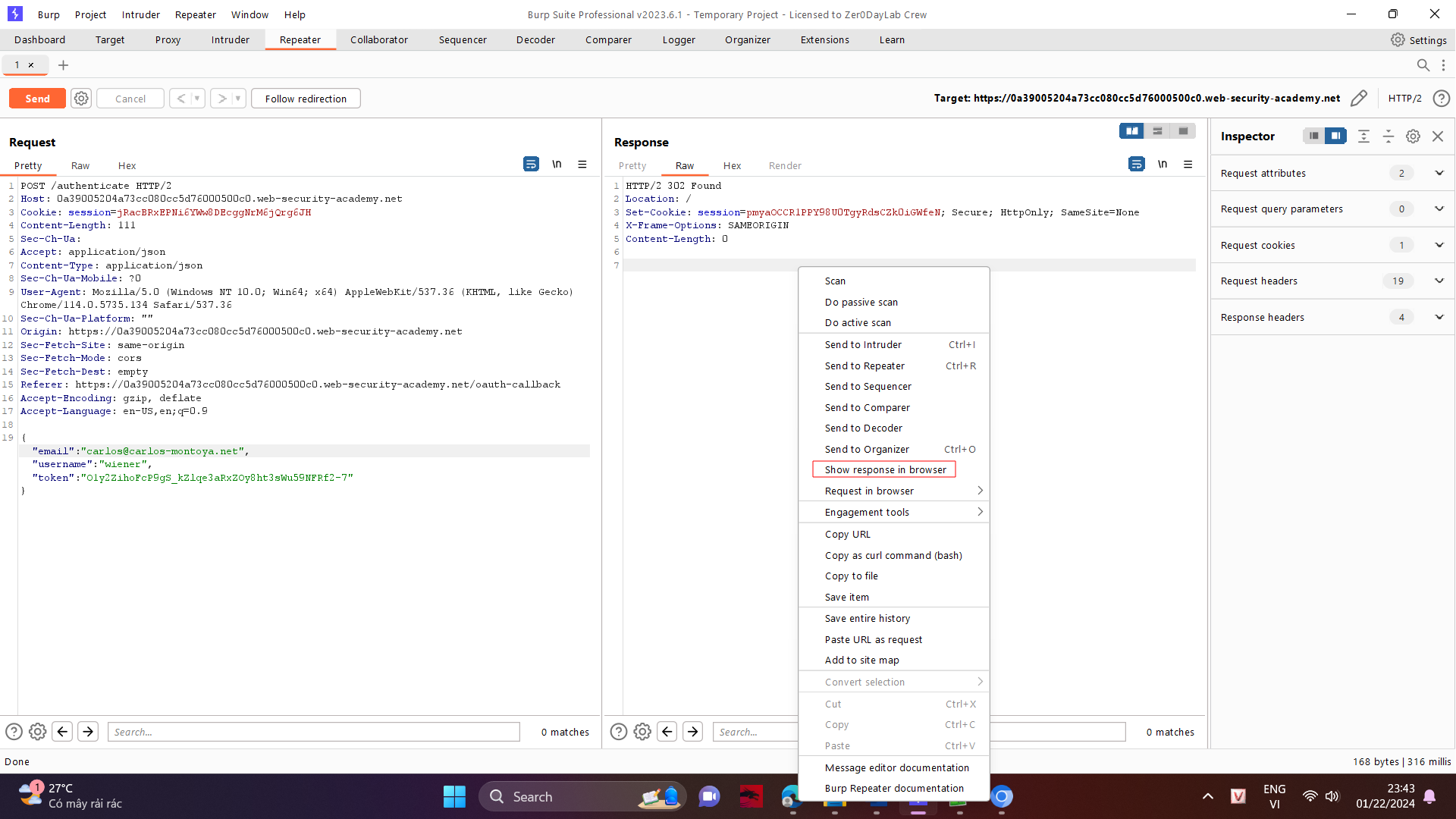


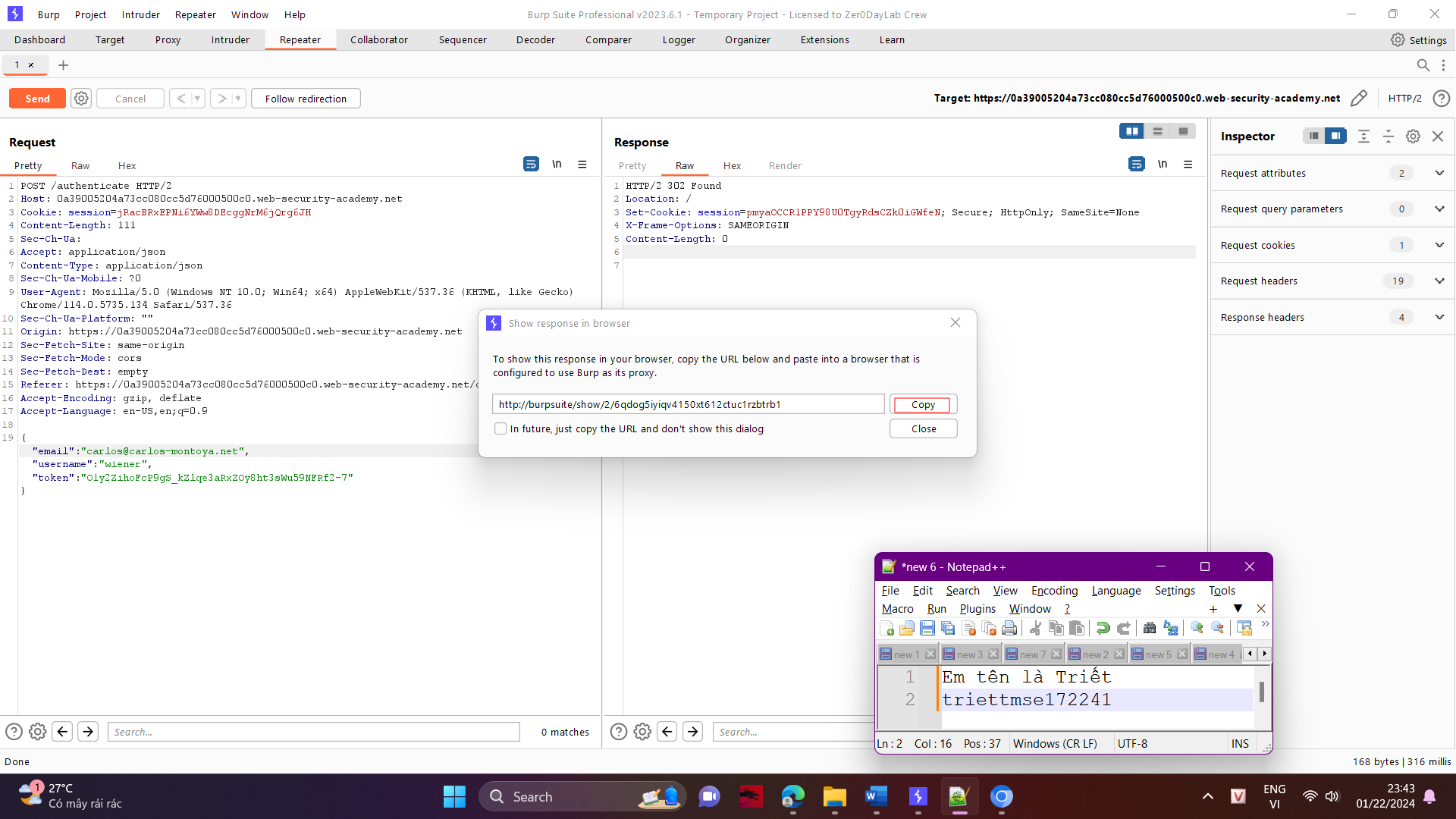
1. Send the POST /authenticate request to Burp Repeater. In Repeater, change the email address to carlos@carlos-montoya.net and send the request. Observe that you do not encounter an error.

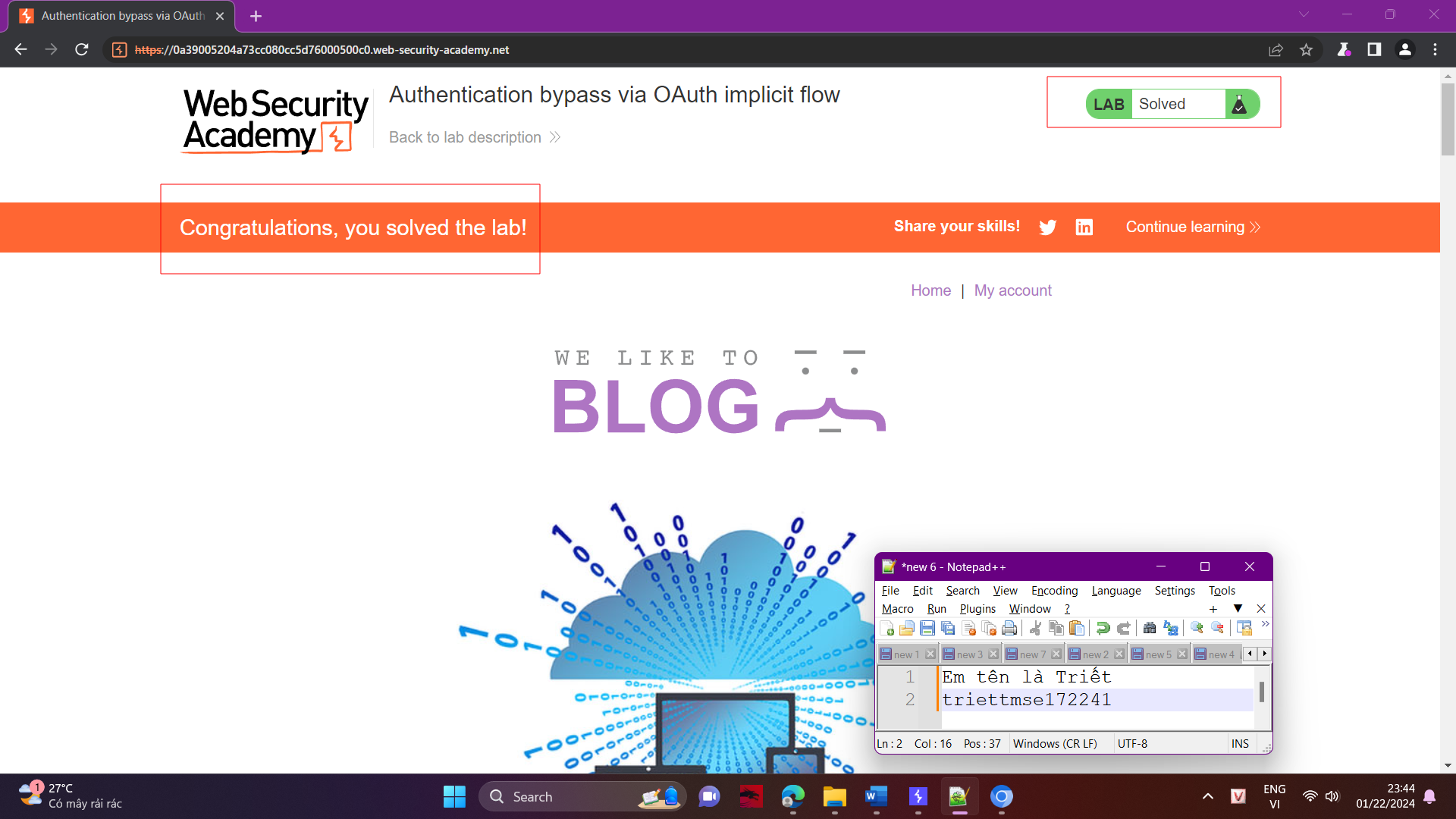


1. Right-click on the POST request and select "Request in browser" > "In original session". Copy this URL and visit it in the browser. You are logged in as Carlos and the lab is solved.









Done!