**QUS 1**

**Vulnerability Assessment Report –** **PortSwigger** **Labs**

# 1. Executive Summary

This report presents a vulnerability assessment performed using PortSwigger Web Security Academy labs. Each lab simulates a real-world scenario that exposes common web application vulnerabilities. The assessment involved identifying, exploiting, and documenting the security flaws to understand their impact and recommend mitigation strategies.

# 2. Scope

This assessment includes five distinct vulnerabilities:  
- SQL Injection (Login Bypass)  
- SQL Injection (Data Retrieval via WHERE Clause)  
- Reflected Cross-Site Scripting (XSS)  
- Server-Side Request Forgery (SSRF)  
- Username Enumeration via Authentication Response Differences  
Tools used include Burp Suite (Community Edition), Firefox Browser, and Kali Linux virtual machine.

# 3. Methodology

The assessment followed a structured approach:  
1. Identification: Understanding the application behavior and locating potential inputs.  
2. Exploitation: Crafting payloads and sending requests to exploit vulnerabilities.  
3. Verification: Confirming exploitation through visible changes or server responses.  
4. Documentation: Capturing screenshots and logging detailed observations.  
5. Recommendation: Proposing mitigation strategies for each vulnerability.

# 4. Findings and Analysis

## SQL Injection – Login Bypass

Objective: Exploit an SQL injection vulnerability in the login form to bypass authentication.

Tools Used: Burp Suite (Proxy & Intercept)

Methodology:

* - Intercepted the login request using Burp Suite.
* - Injected a malicious SQL payload `' OR 1=1--` into the username field.
* - Submitted the modified request.

Result: Successfully bypassed login and accessed the administrator account.

Recommendation: Use parameterized queries and input validation to prevent SQL injection.

Screenshots:

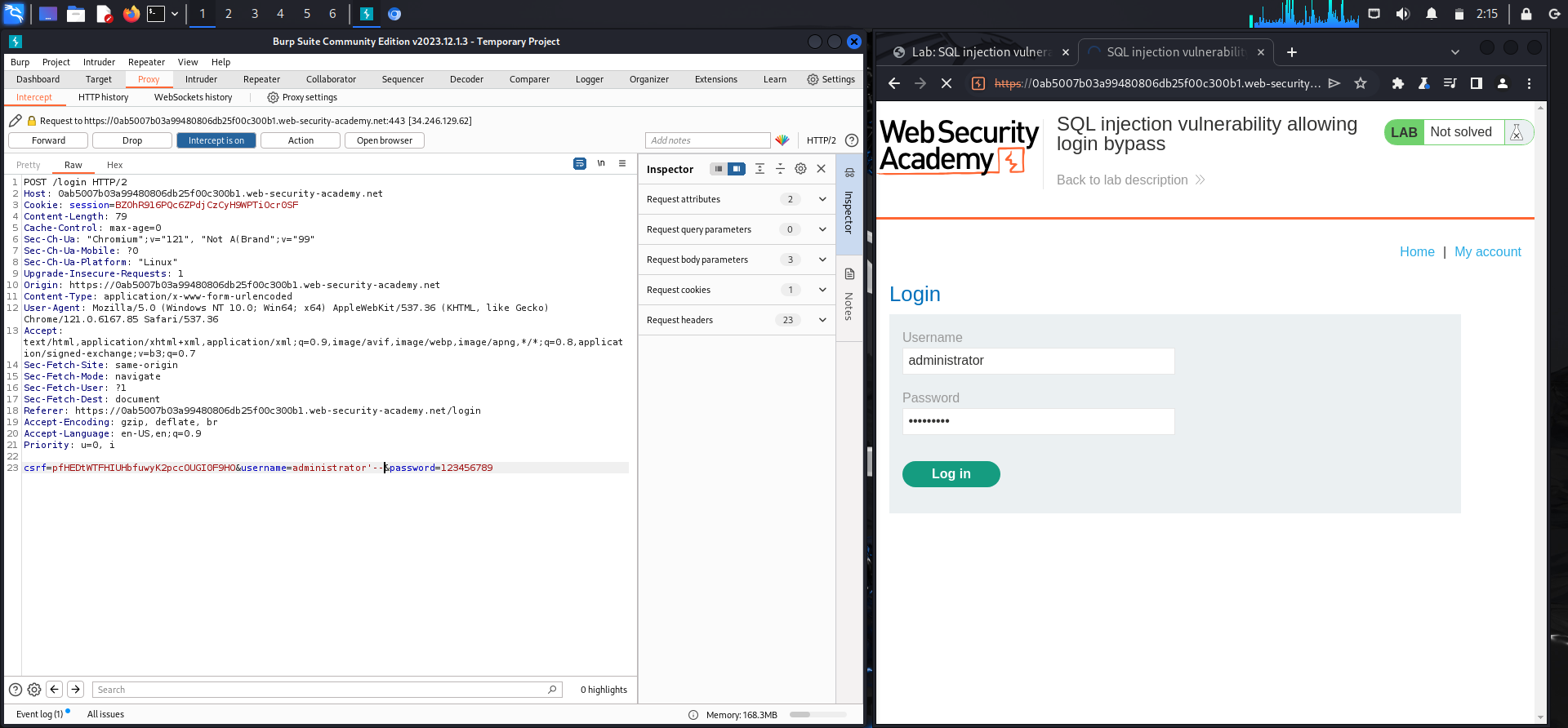


Figure: SQL INJ login bypass.png

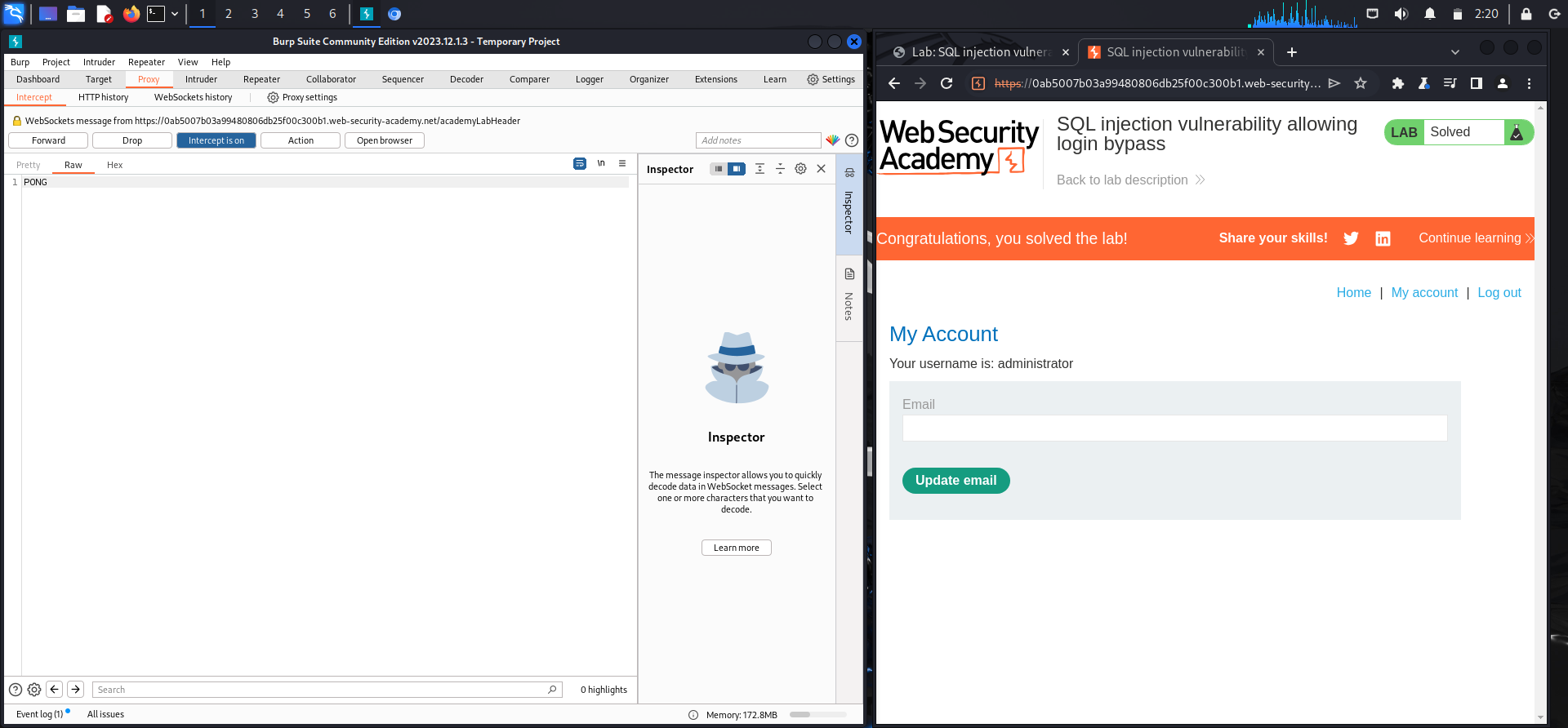


Figure: SQL INJ login bypass2.png

## SQL Injection in WHERE Clause (Retrieval of Hidden Data)

Objective: Extract hidden product data from the database by injecting into the WHERE clause.

Tools Used: Burp Suite (Repeater)

Methodology:

* - Located a category filter parameter vulnerable to injection.
* - Injected the payload `Tech gifts' OR 1=1--` into the parameter.
* - Sent the request using Burp Repeater and analyzed the response.

Result: Successfully retrieved all products, bypassing the category filter.

Recommendation: Sanitize user inputs and use ORM frameworks or prepared statements.

Screenshots:

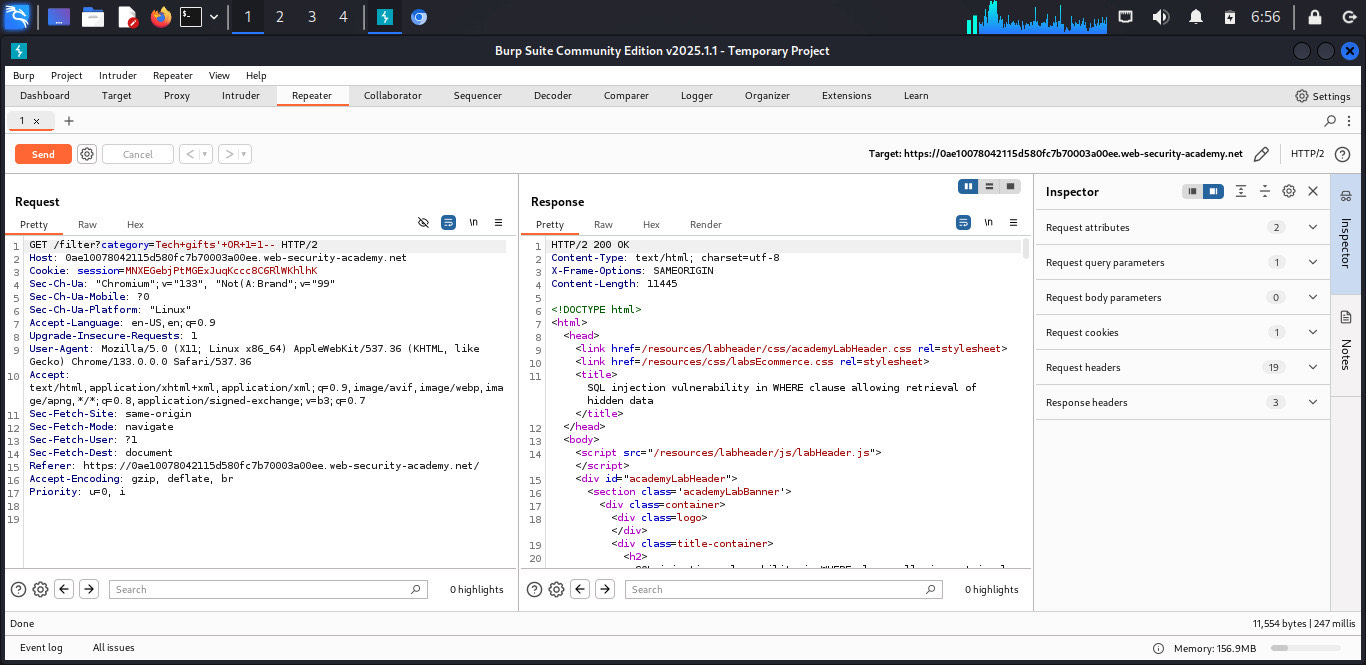


Figure: SQL INJECTION retrieval of HD.jpeg

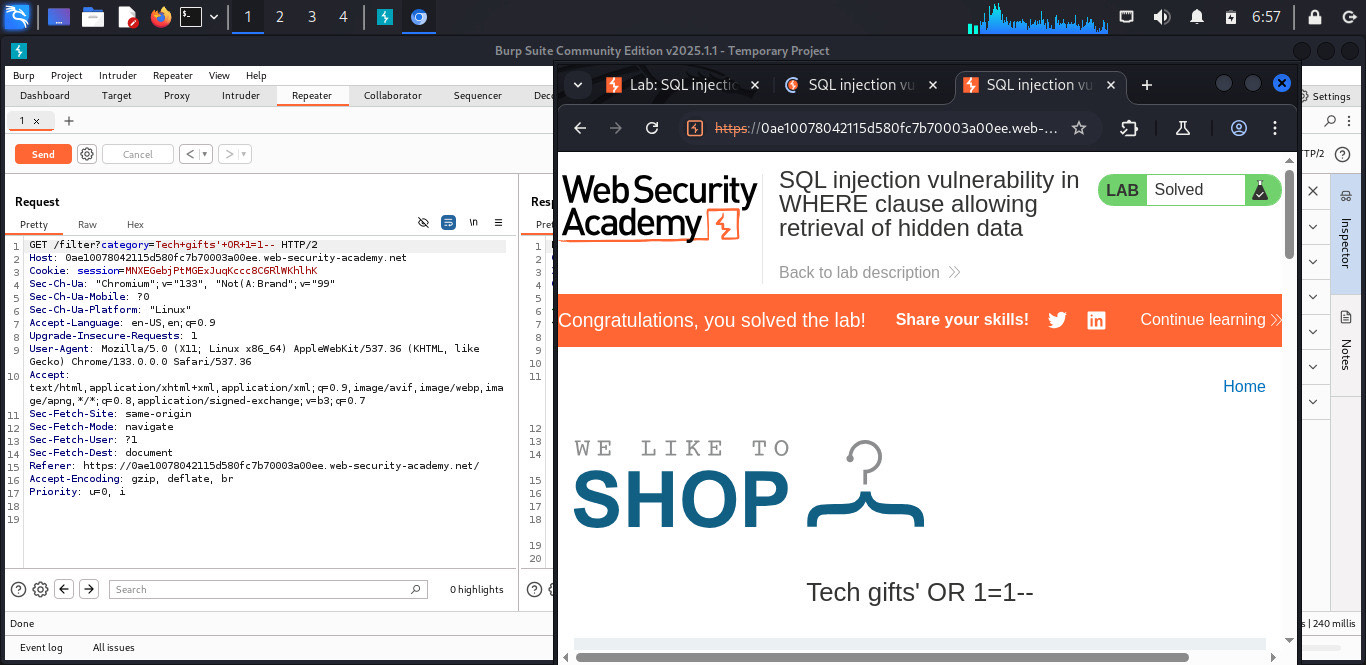


Figure: SQL INJECTION retrieval of HD2.jpeg

## Cross-Site Scripting (Reflected XSS)

Objective: Execute arbitrary JavaScript using reflected XSS.

Tools Used: Browser + Burp Suite

Methodology:

* - Identified the input field that reflected user input in the response.
* - Injected a payload `<script>alert(1)</script>`.
* - Observed script execution in browser.

Result: The script was executed successfully, proving a reflected XSS vulnerability.

Recommendation: Sanitize and encode user input in output. Use Content Security Policy (CSP).Screenshots:

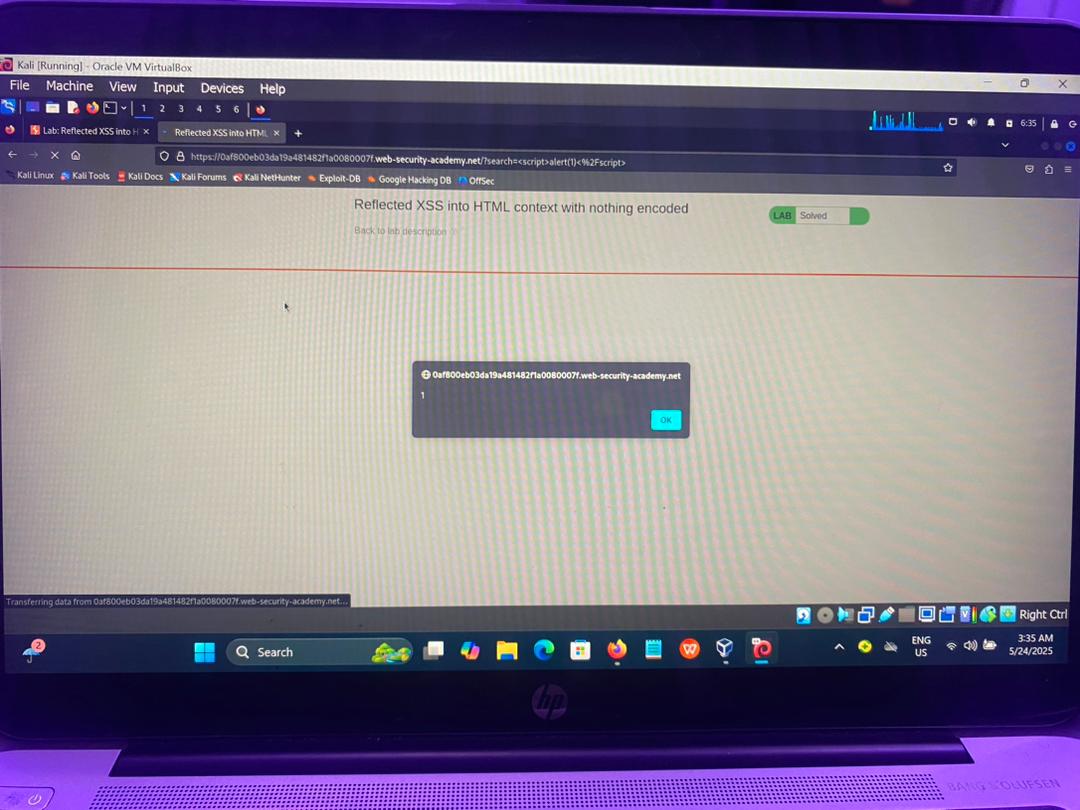


Figure: WhatsApp Image 2025-05-24 at 3.36.47 AM.jpeg

## Server-Side Request Forgery (SSRF)

Objective: Exploit SSRF to make the server access internal resources.

Tools Used: Burp Suite (Repeater)

Methodology:

* - Located a product stock checker endpoint vulnerable to SSRF.
* - Modified the `stockApi` parameter to target `http://localhost/admin/delete?username=carlos`.
* - Sent the modified request through Burp Repeater.

Result: The request deleted the specified user, confirming SSRF exploitation.

Recommendation: Validate and restrict URLs in SSRF-prone parameters. Use allowlists.

Screenshots:

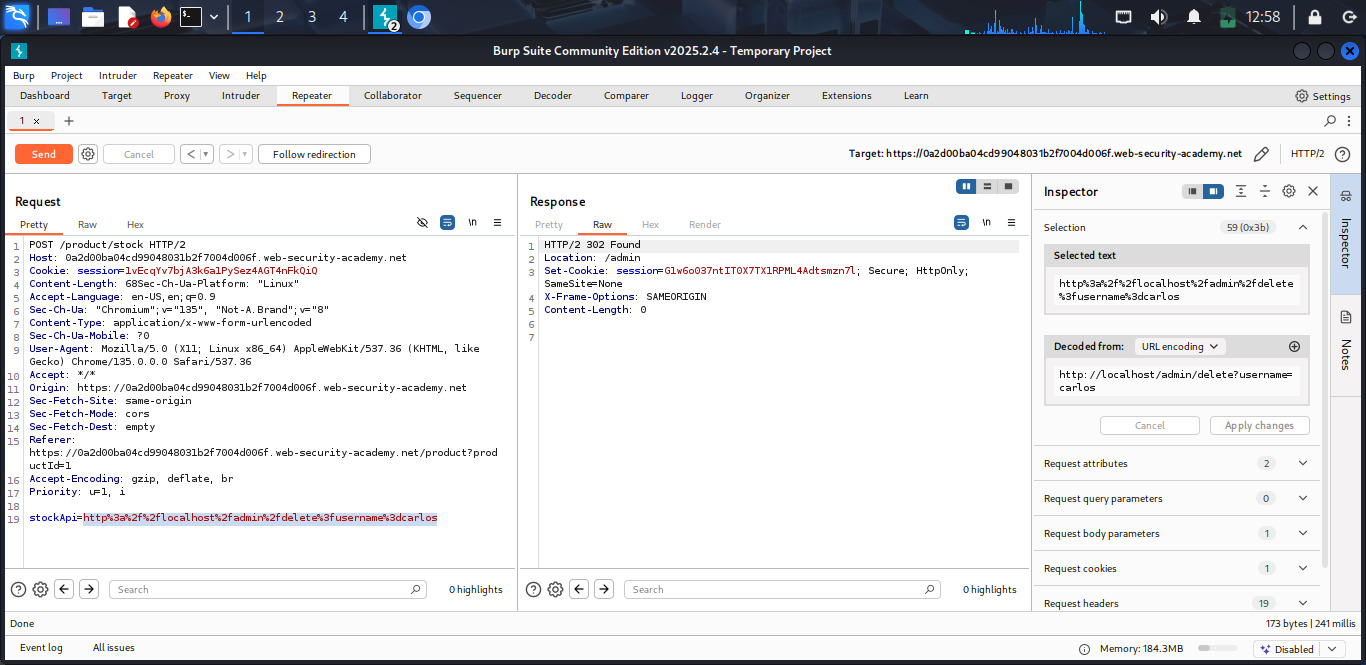


Figure: WhatsApp Image 2025-05-16 at 10.04.29 AM(1).jpeg

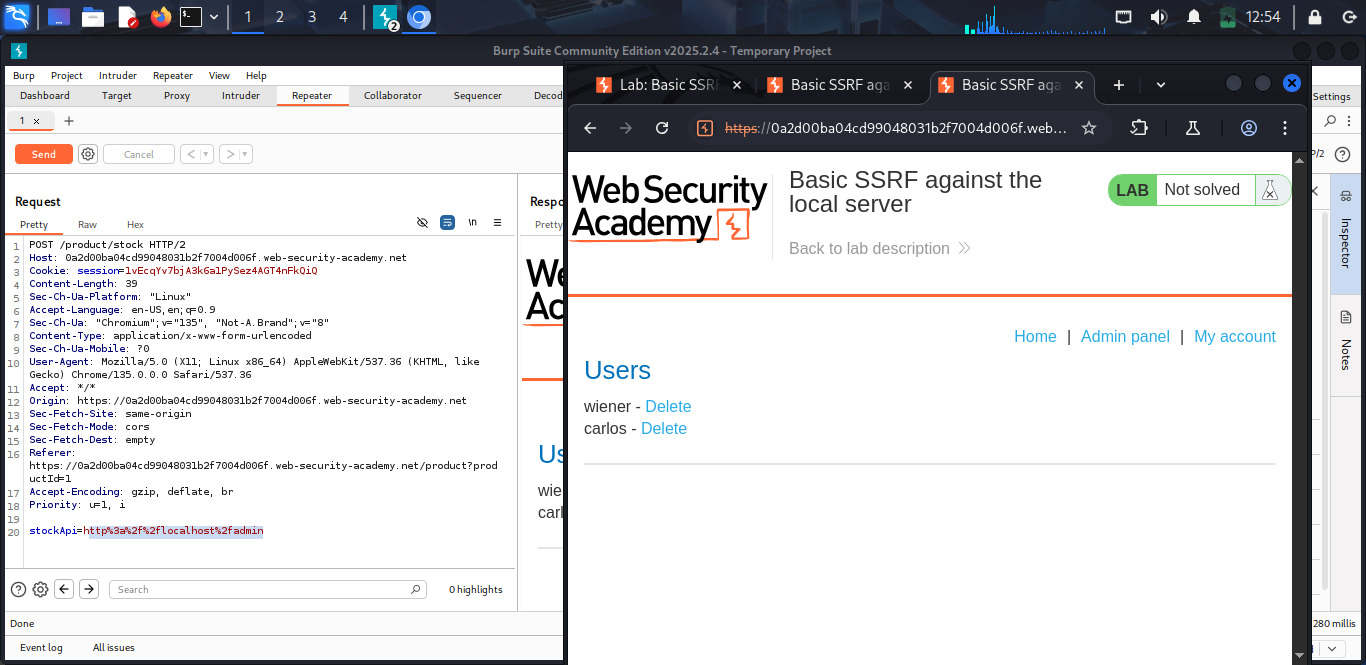


Figure: WhatsApp Image 2025-05-16 at 10.04.29 AM(2).jpeg

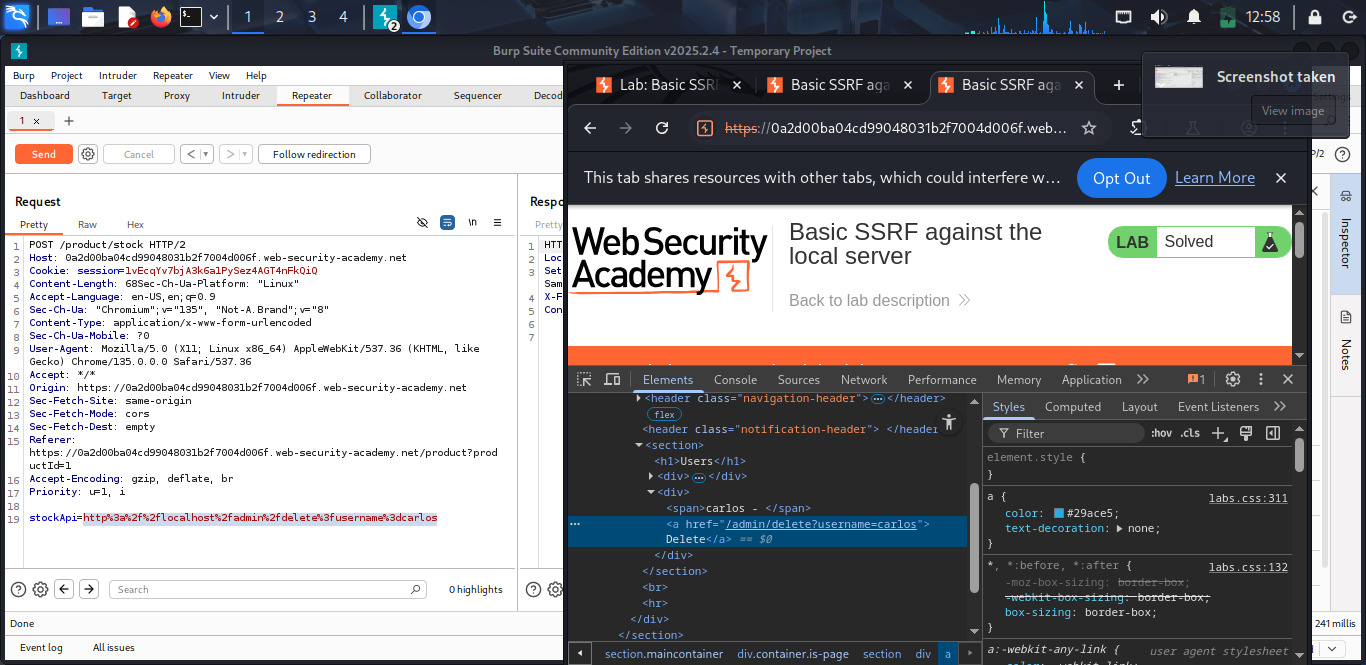
Z

Figure: WhatsApp Image 2025-05-16 at 10.04.29 AM.jpeg

## Authentication Vulnerability (Username Enumeration)

Objective:

Identify valid usernames based on response differences.

Tools Used: Burp Suite (Intruder)

Methodology:

* - Configured Burp Intruder to send a list of usernames to the login endpoint.
* - Monitored response lengths and messages to find anomalies.
* - Detected valid usernames from different response behaviors.

Result: Valid username (`appserver`) identified via different response lengths and messages.

Recommendation: Standardize authentication error messages and implement rate limiting.

Screenshots:

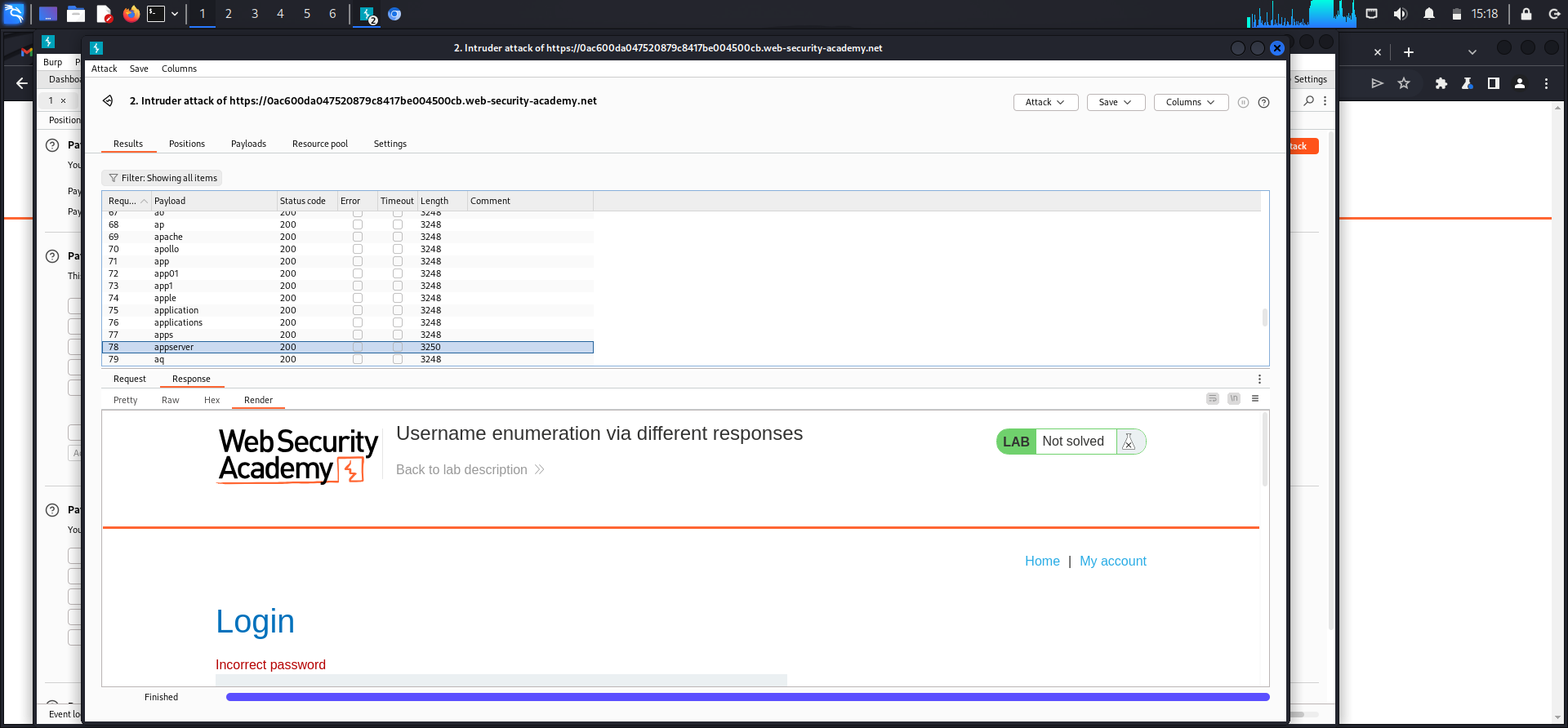


Figure: Screenshot\_2025-05-26\_15-18-54.png

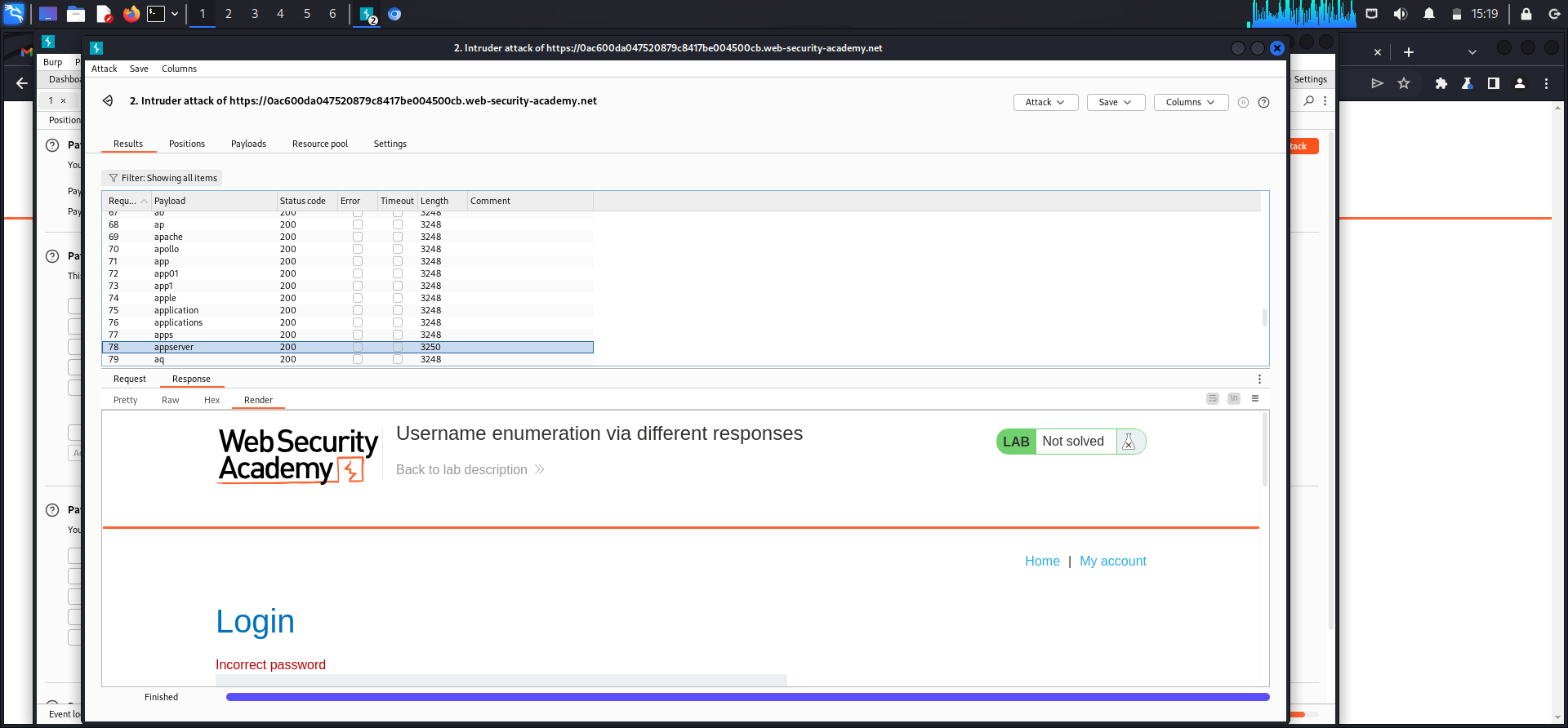


Figure: Screenshot\_2025-05-26\_15-19-09.png

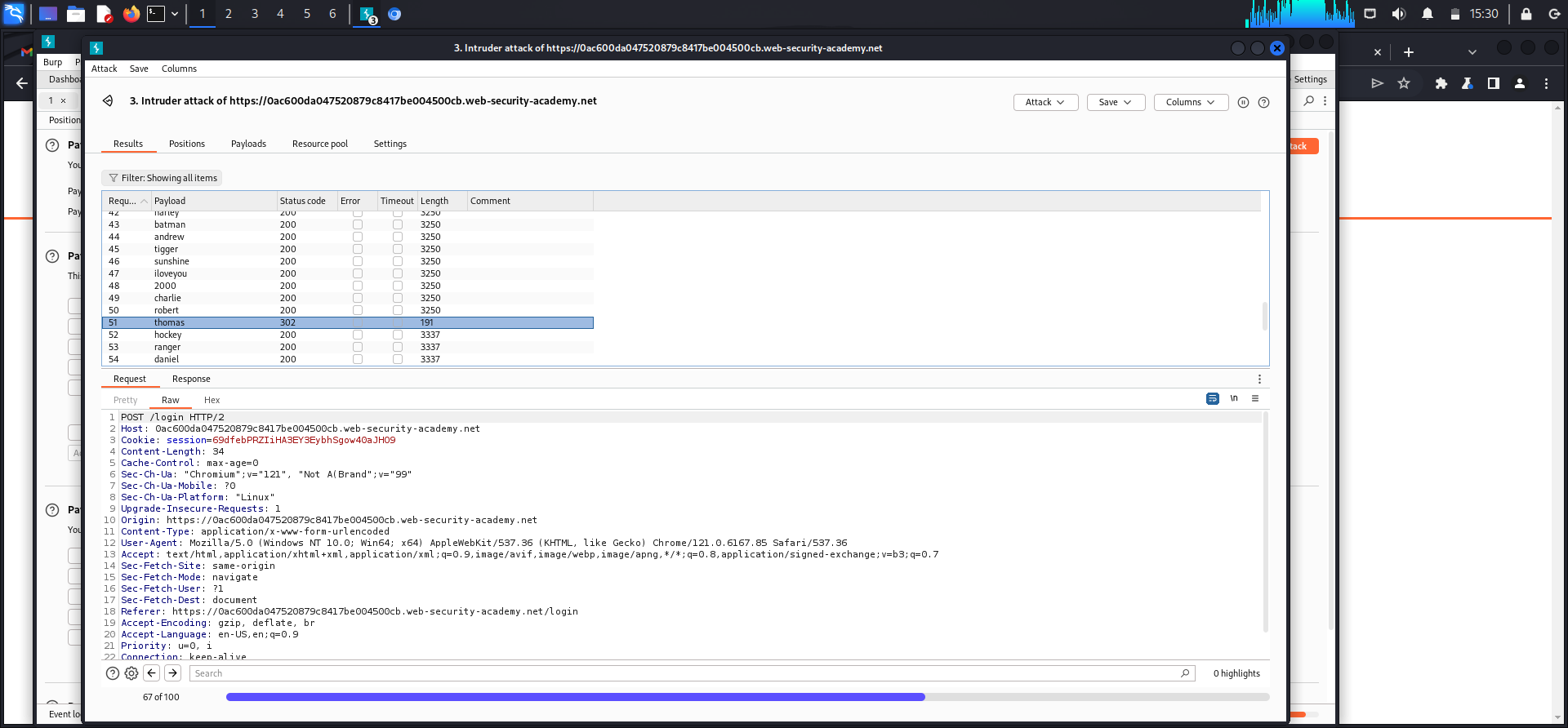


Figure: Screenshot\_2025-05-26\_15-30-18.png



Figure: Screenshot\_2025-05-26\_15-32-51.png

# 5. Conclusion

This assessment reinforces the critical need for secure coding practices in web development. Each vulnerability demonstrated how attackers can exploit even basic flaws to gain unauthorized access, extract sensitive data, or interfere with server operations. Organizations must adopt secure development lifecycles, conduct regular penetration testing, and implement robust input validation and output encoding techniques.

**QUS 2**

**Using NMAP To Run a Vulnerability Scan On The Devices on The Lab**

**1 .** System overview

Target: Kali Linux Virtual Machine

Date of Scan: June 18, 2025

Scanner Used: Nmap v7.94SVN

Command Executed: sudo nmap -sV --script vuln 10.0.2.15

This scan was conducted on a Kali Linux machine operating in a VirtualBox environment. The purpose was to

assess the system for any running services and detect potential vulnerabilities. The scan targeted the

system's local IP address (10.0.2.15), focusing specifically on identifying version information and known

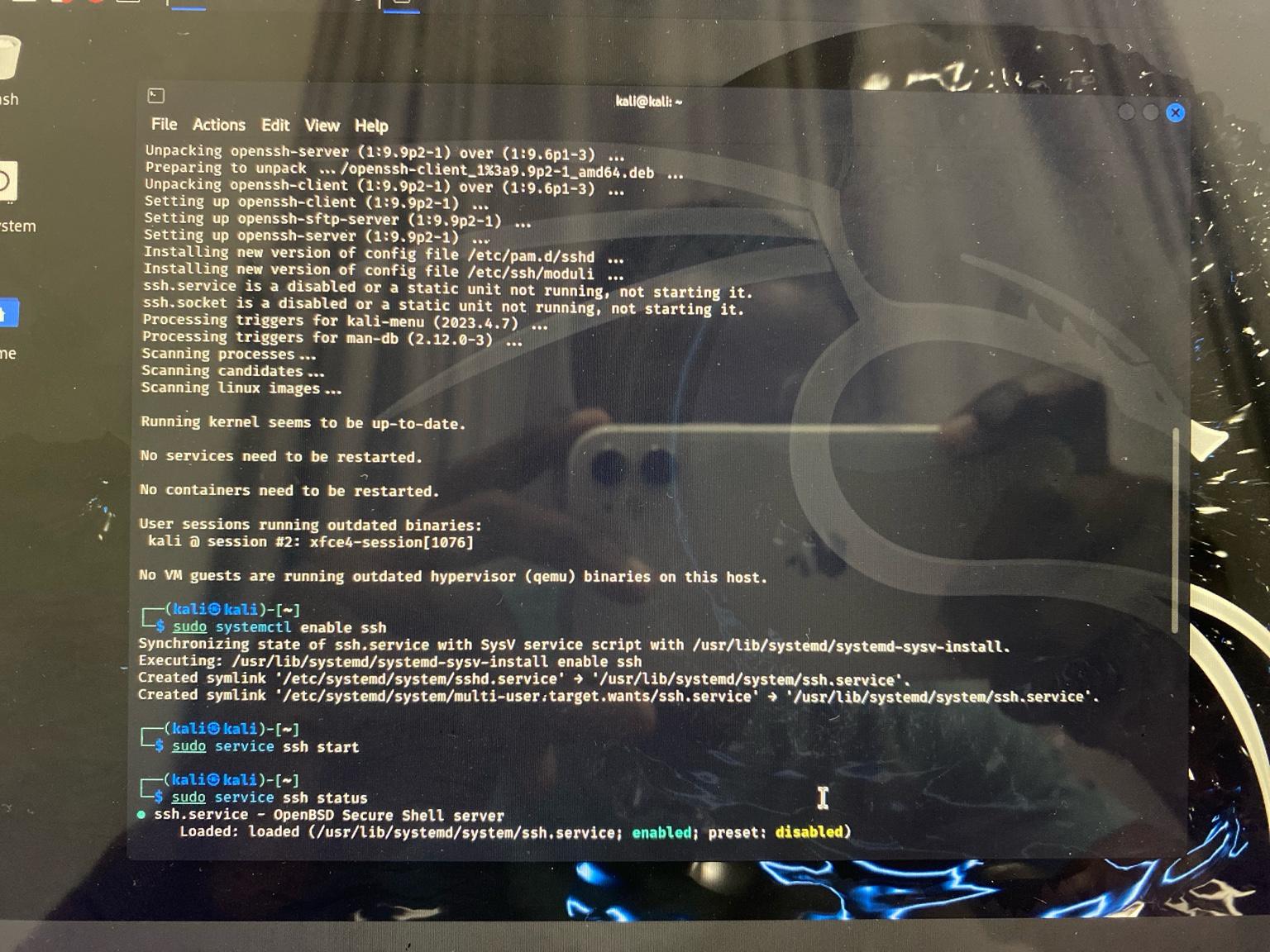
vulnerabilities using Nmap's scripting engine.

Primary findings indicate the SSH (Secure Shell) service was the only service running and accessible through

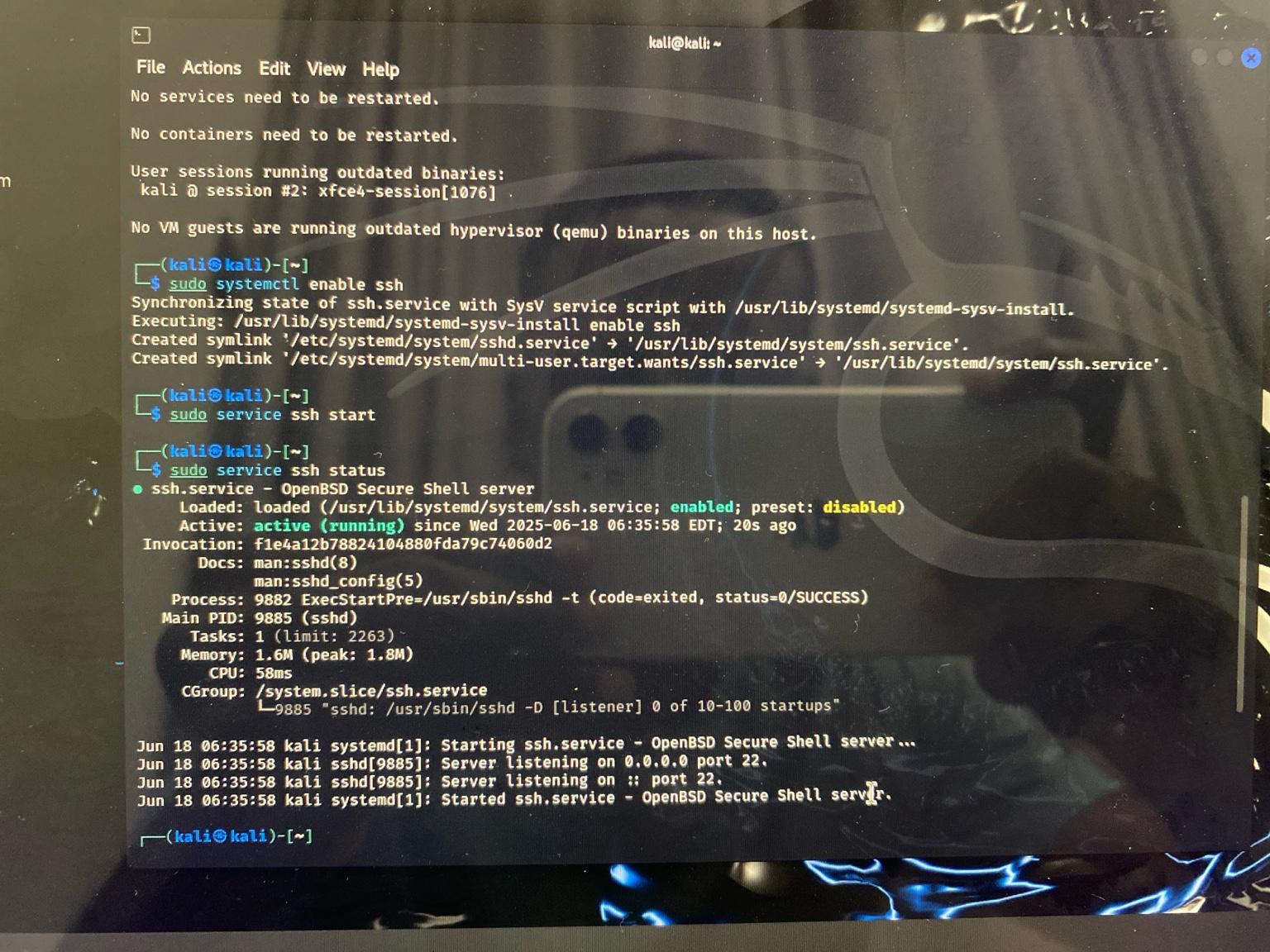
port 22. This service allows remote command-line access and is commonly targeted in penetration tests due

to its potential exposure to brute-force attacks, misconfigurations, or software vulnerabilities.

1. Below are screenshots showing the process of enabling and verifying the SSH service on the machine.



*Figure 1: Enabling SSH on Kali Linux*



*Figure 2: Verifying SSH is active and listening on port 22*

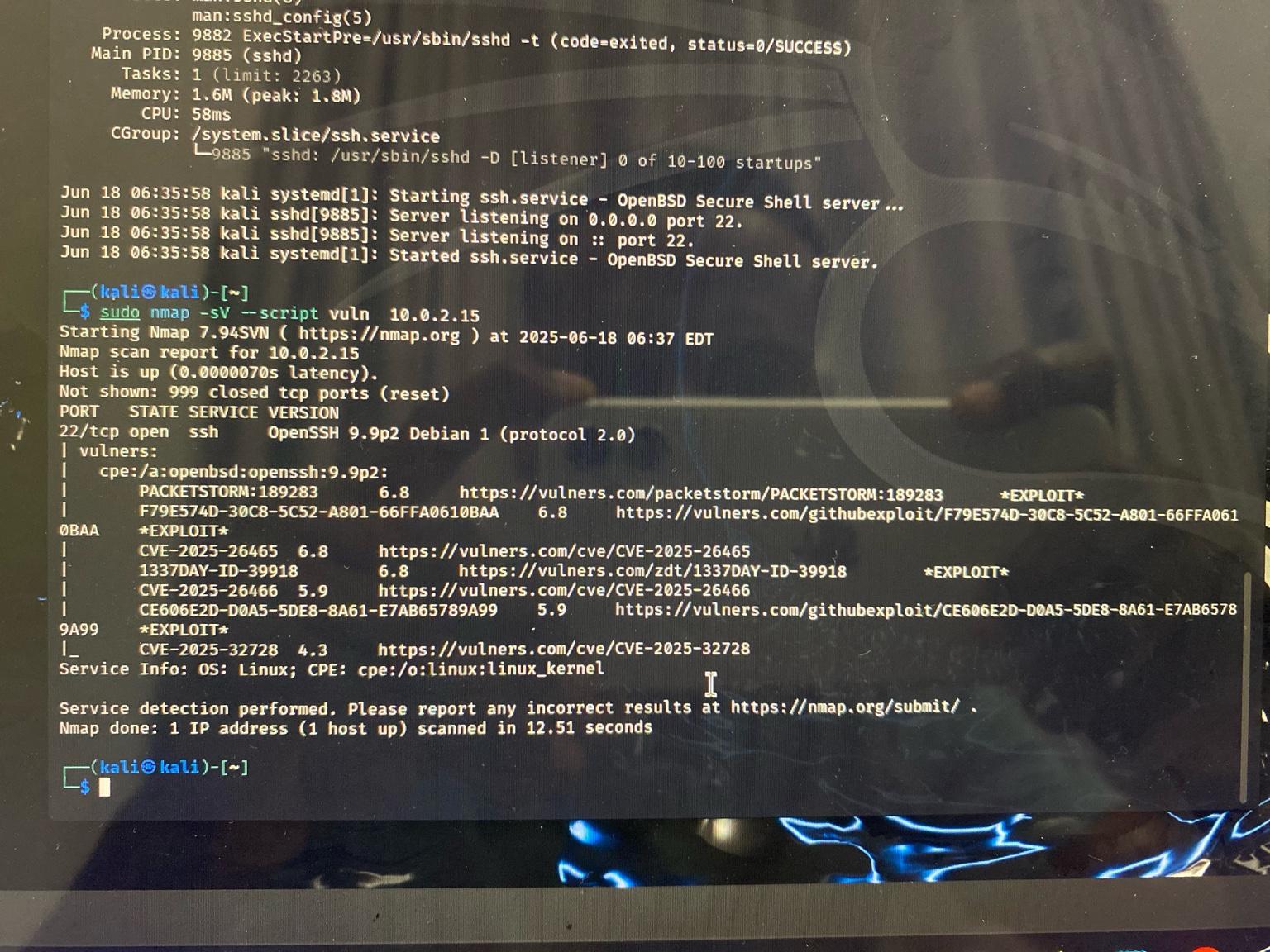
# Vulnerability Scan

Once SSH was confirmed to be running, an Nmap scan was conducted targeting port 22 using the `--script vuln` flag. This script queries known vulnerabilities associated with the service and software version detected.

The screenshot below shows the scan result, which confirms that port 22 is open and running OpenSSH

9.9p2 on Debian. Several known vulnerabilities and publicly available exploits were detected during the scan. These include kernel-related issues and OpenSSH-specific flaws.

The vulnerabilities are linked to CVEs like CVE-2025-3086 and CVE-2025-32728, which could potentially allow privilege escalation or unauthorized access if exploited.



*Figure 3: Nmap vulnerability scan on SSH service (port 2*

3 . **Recommendations**

To reduce the risk of exploitation, the following remediation steps are recommended:

1. Update OpenSSH and Kernel: Apply patches to resolve the listed CVEs.
2. Use Key-Based Authentication: This is more secure than password authentication and eliminates brute-force risks.
3. Restrict SSH Access: Limit access using a firewall and allow only specific IPs to connect.
4. Disable Root Login Over SSH: Modify the SSH config file to prevent direct root login.
5. Regular Monitoring: Set up a schedule for regular vulnerability assessments and apply updates promptly.

These steps ensure the system remains secure even if exposed to an external network.

1. Conclusion

The assessment shows that while SSH is successfully enabled and functioning on the Kali machine, it is vulnerable due to known issues in the OpenSSH version and underlying Linux kernel. These issues were confirmed using Nmap's vulnerability script.

By applying the suggested recommendations, the system can be hardened against external threats, ensuring a safer environment for remote access and administrative control.

**QUS 3**

**Phishing** **Simulation Project Report – GoPhish Tool**

# 1. Introduction

This report details a phishing simulation carried out using the GoPhish open-source phishing framework. The objective of this exercise was to test the delivery and effectiveness of phishing emails, assess user interaction, and evaluate the potential risks of social engineering attacks through email-based vectors.

# 2. Tool Setup: GoPhish

GoPhish was installed and accessed via localhost (`http://127.0.0.1:3333`). The setup included creating user groups, configuring email templates, designing a phishing landing page, and deploying a campaign targeting test accounts. The landing page used was a cloned version of Google’s login page, with options to capture submitted data and passwords.

Screenshot – Landing Page Setup:

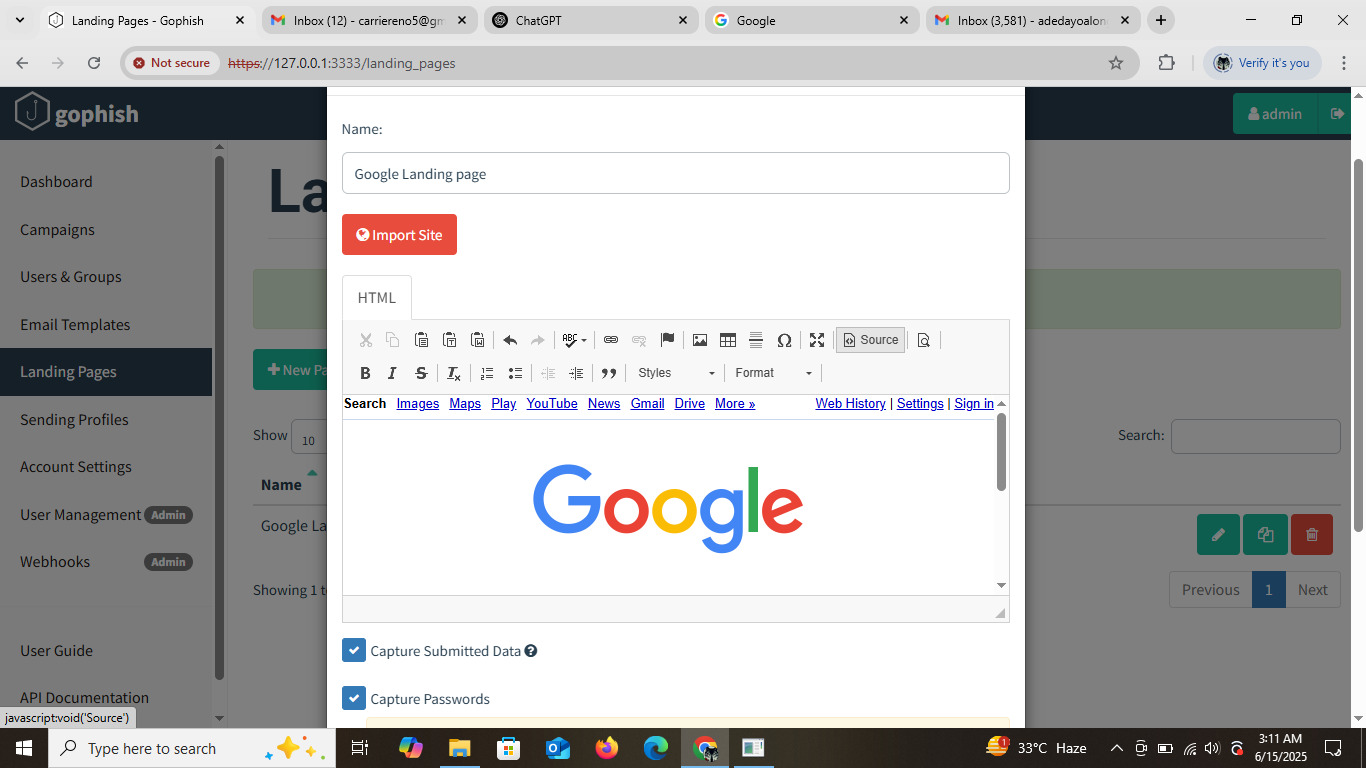


Figure: Custom phishing landing page mimicking Google, designed to capture login credentials.

# 3. Campaign Execution

A phishing campaign was launched targeting two controlled email addresses owned by the tester: `donaldchris002@gmail.com` and `adedayoalonge71@gmail.com`. The phishing email contained a subject line related to account security and a link directing the user to the fake Google login page. The goal was to measure engagement (opens, clicks) without capturing actual credentials for ethical compliance.

Screenshot – Phishing Email in Spam Folder:



Figure: The phishing email landed in the spam folder with a message urging the user to verify their account due to suspicious activity.

# 4. Results and Findings

The GoPhish dashboard provided real-time analytics for the campaign. The summary of results is as follows:  
- Emails Sent: 2  
- Emails Opened: 2  
- Links Clicked: 2  
- Data Submitted: 0  
- Emails Reported: 0

Screenshot – GoPhish Campaign Dashboard:

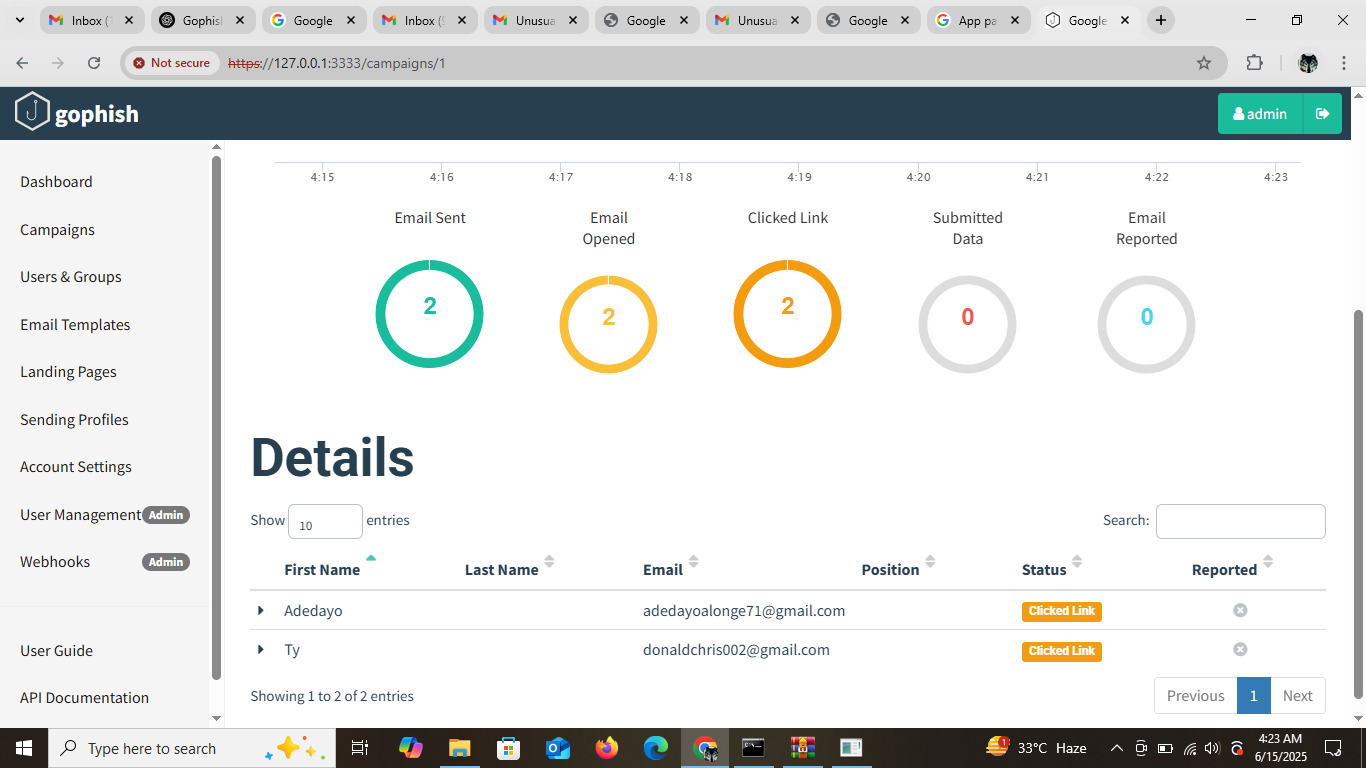


Figure: Dashboard showing full interaction from both users — both emails were opened and links clicked.

# 5. Analysis

The phishing campaign successfully demonstrated typical user behavior in a phishing scenario:  
- High open and click rate (100%) showed users are likely to engage with well-crafted phishing emails.  
- The email was flagged as spam, indicating Gmail’s filters are effective but still allow user access.  
- No credentials were submitted, likely due to lack of interaction or awareness by the tester.  
  
This simulation confirms the risk posed by phishing attacks and the necessity of user training and robust email filtering.

# 6. Recommendations

- Implement advanced email filtering and spam detection rules.  
- Educate users about phishing red flags and verification procedures.  
- Use email authentication protocols such as SPF, DKIM, and DMARC.  
- Regularly conduct phishing simulations to raise awareness.

# 7. Conclusion

The GoPhish-based simulation demonstrated how phishing emails can bypass spam filters and prompt user interaction. While no sensitive data was captured in this ethical test, it highlights the importance of continuous vigilance, security education, and layered defense strategies against phishing threats in real-world scenarios.

**QUS 4**

**Log Analysis Report: Event ID 4625 (Failed Logon)**

1. Objective

This report provides an analysis of Event ID 4625 (Failed Logon Attempt) captured in the Windows Security log.

The objective is to understand the cause of the failed logon, evaluate potential risks, and recommend appropriate security controls.

1. Event Summary Event ID: 4625 Date & Time: 06/23/2025 3:29:46 AM Source:

Microsoft-Windows-Security-Auditing Logon Type: 2 (Interactive) Authentication Package: Negotiate Logon Process: User32 Failure Reason: Unknown user name or bad password

Workstation Name: DESKTOP-FSSK95F Source IP: 127.0.0.1 (localhost) Process Name: C:\Windows\System32\svchost.exe Account Name Tried: hp

1. Event Description and Interpretation Event ID 4625 is logged whenever a logon attempt fails:

In this case, the logon type was 2, which indicates an interactive (local console) login attempt.

The account name used was 'hp', and the error suggests either the username doesn't exist or the password was incorrect. The logon attempt originated from localhost (127.0.0.1), meaning it was triggered directly from the machine and not remotely.

The use of 'svchost.exe' implies that the attempt could be tied to a system-level background service trying to authenticate or possibly even a user-initiated login via service-dependent mechanisms.

1. Security Implications Although this single failed attempt might not indicate a threat, repeated or patterned failed logon attempts-especially from privileged services like svchost.exe-can be signs of brute-force attacks or malware behavior. Failed logins from system accounts should be carefully monitored to detect internal misuse or misconfiguration.
2. Security Controls to Implement

1. Enforce Account Lockout Policy: Lock accounts after 5 failed logon attempts for 15-30 minutes.

2. Implement Multi-Factor Authentication (MFA) for local and remote users.

3. Monitor svchost.exe and other system processes for unusual authentication attempts.

4. Regularly review audit logs using centralized tools (e.g., Wazuh, Splunk).

5. Use alerts for repeated Event ID 4625, especially during off-hours.

6. Educate users on strong password hygiene and ensure no test or unused accounts are active.

1. Conclusion

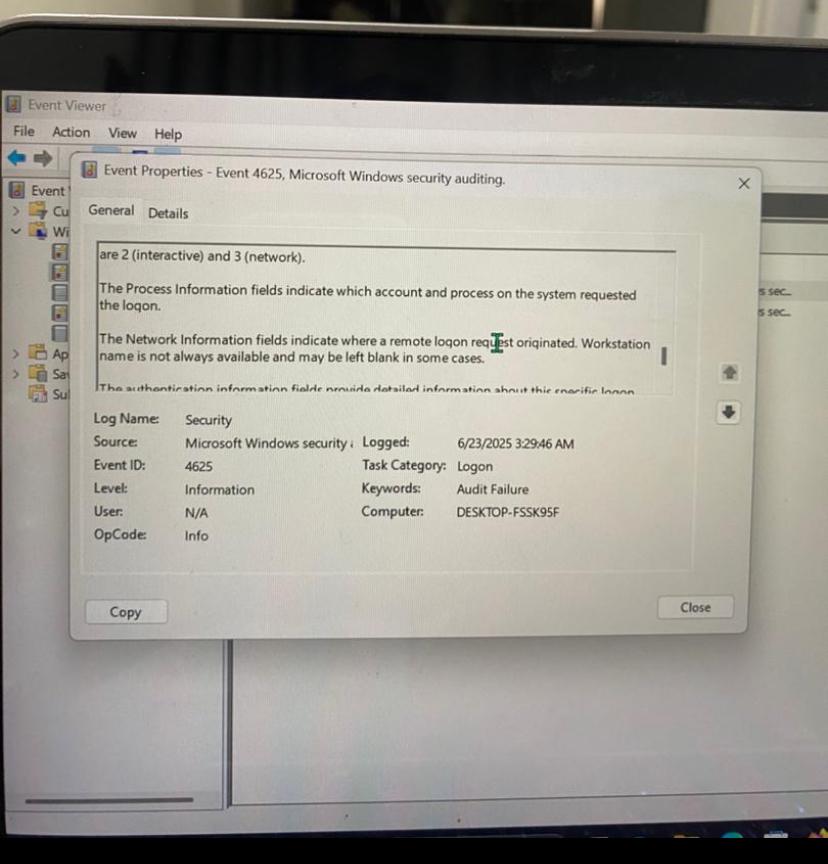
This log analysis of Event ID 4625 reveals a failed interactive logon attempt using an invalid username or password. The local origin and use of svchost.exe are notable, requiring continued monitoring.

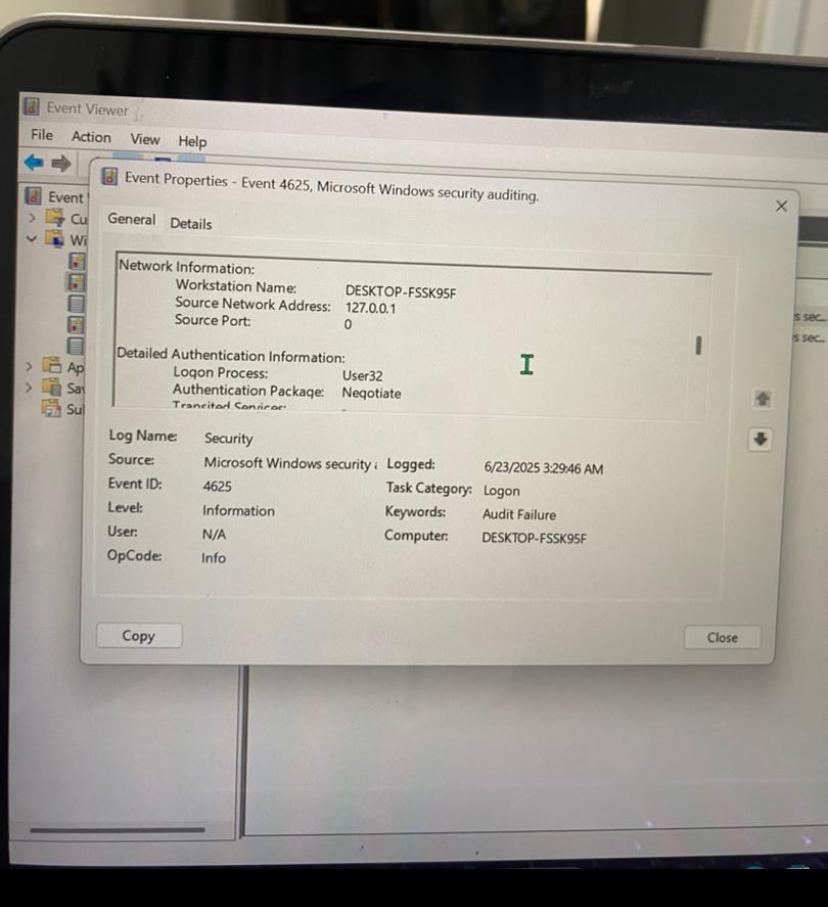
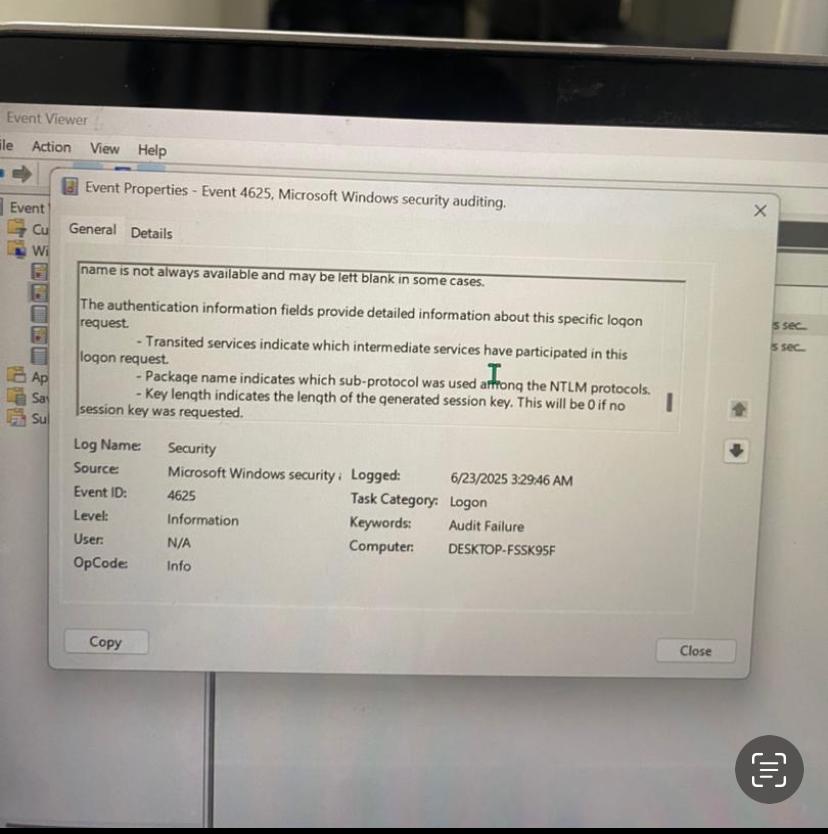
Implementing the recommended security controls will help minimize risk and detect similar anomalies in the future.

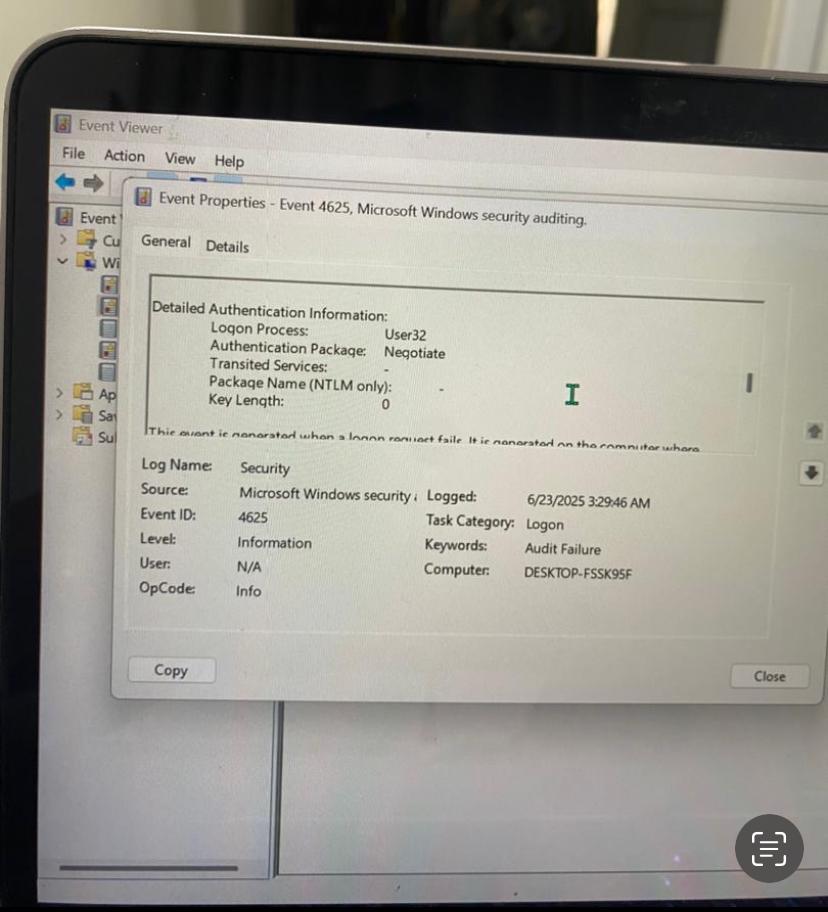
1. Screenshot Illustrations

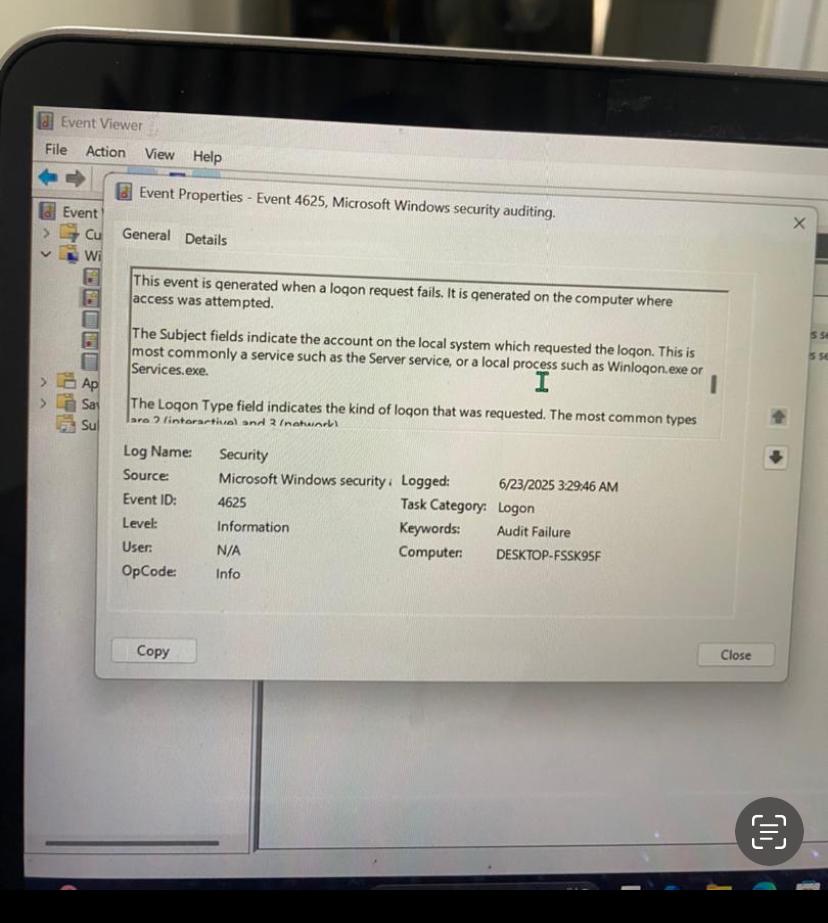
Below are the screenshots captured during the analysis of the failed logon attempt (Event ID 4625) from

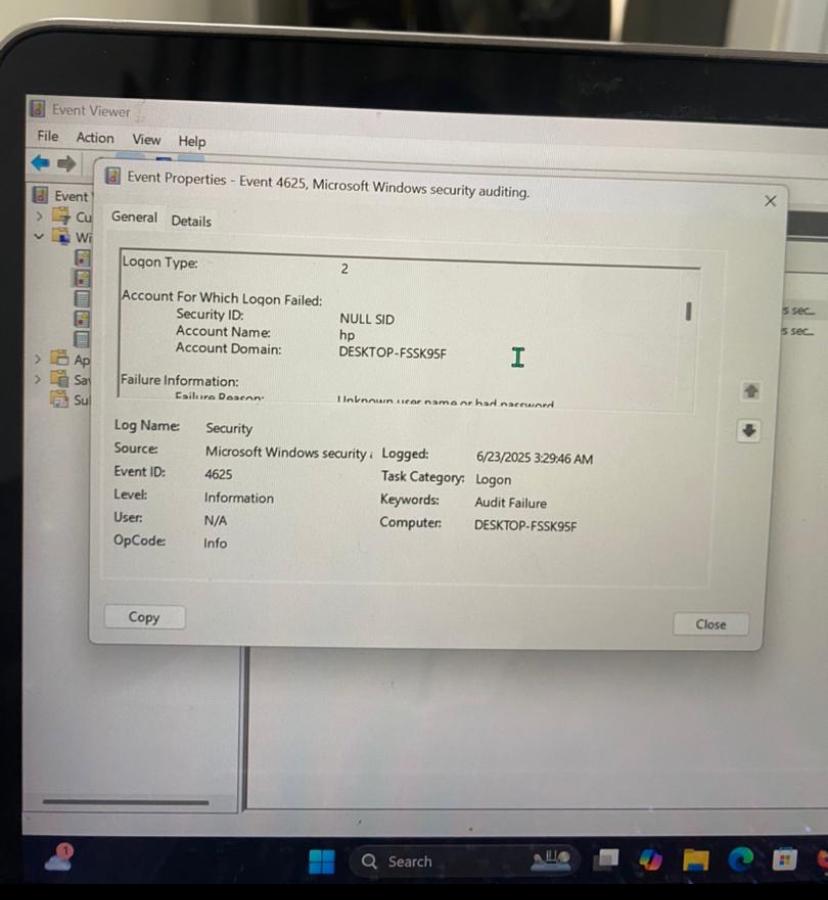
Event Viewer :

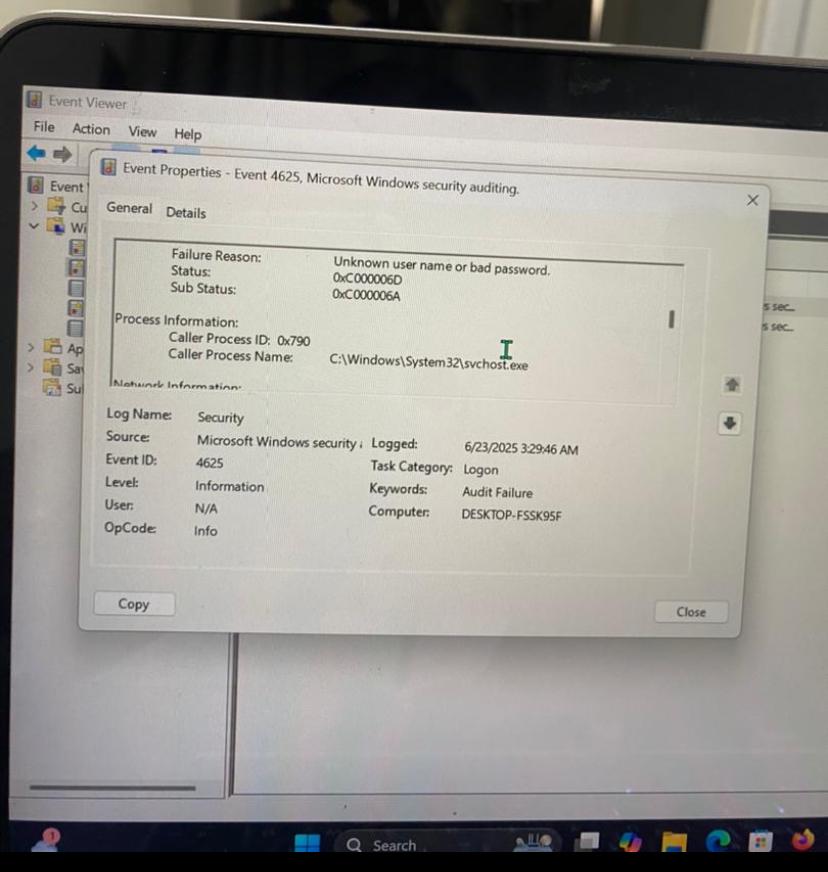


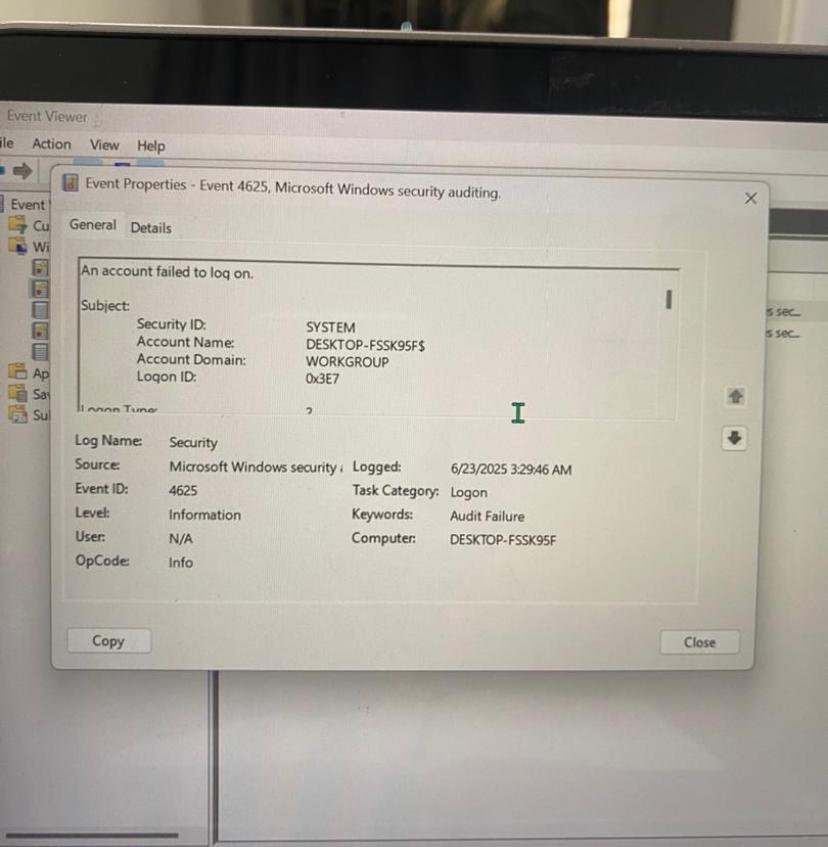






****



ss

QUS 5

Mobile Application Security Testing Report – DVMA

# 1. Introduction

This report summarizes the results of a security assessment conducted on the Damn Vulnerable Mobile Application (DVMA). DVMA is a deliberately insecure mobile app used for learning and testing common Android vulnerabilities. The purpose of this exercise was to identify and understand security flaws in mobile applications using a controlled testing environment.

# 2. Environment Setup

The following tools and configurations were used for testing:  
Emulator: Nexus 5 API 23 (Android Studio)  
-Testing Tool: Android Studio & Emulator  
- DVMA APK: Deployed on emulator  
- Access: Android Debug Bridge (ADB) and Logcat monitoring.

# 3. Vulnerabilities Tested and Findings

## 3.1 Insecure Logging

Insecure logging occurs when sensitive data (e.g., passwords, credit card numbers) is logged to the system logs. This information can be accessed by attackers via logcat or if the device is rooted.  
  
Objective: Identify if sensitive data is logged.  
Method: Input credit card number in the test field and observe logcat output.  
Result: Credit card data was printed in plain text in the log (visible in terminal).  
Recommendation: Avoid logging sensitive data. Use conditional logging for debugging and remove in production builds.

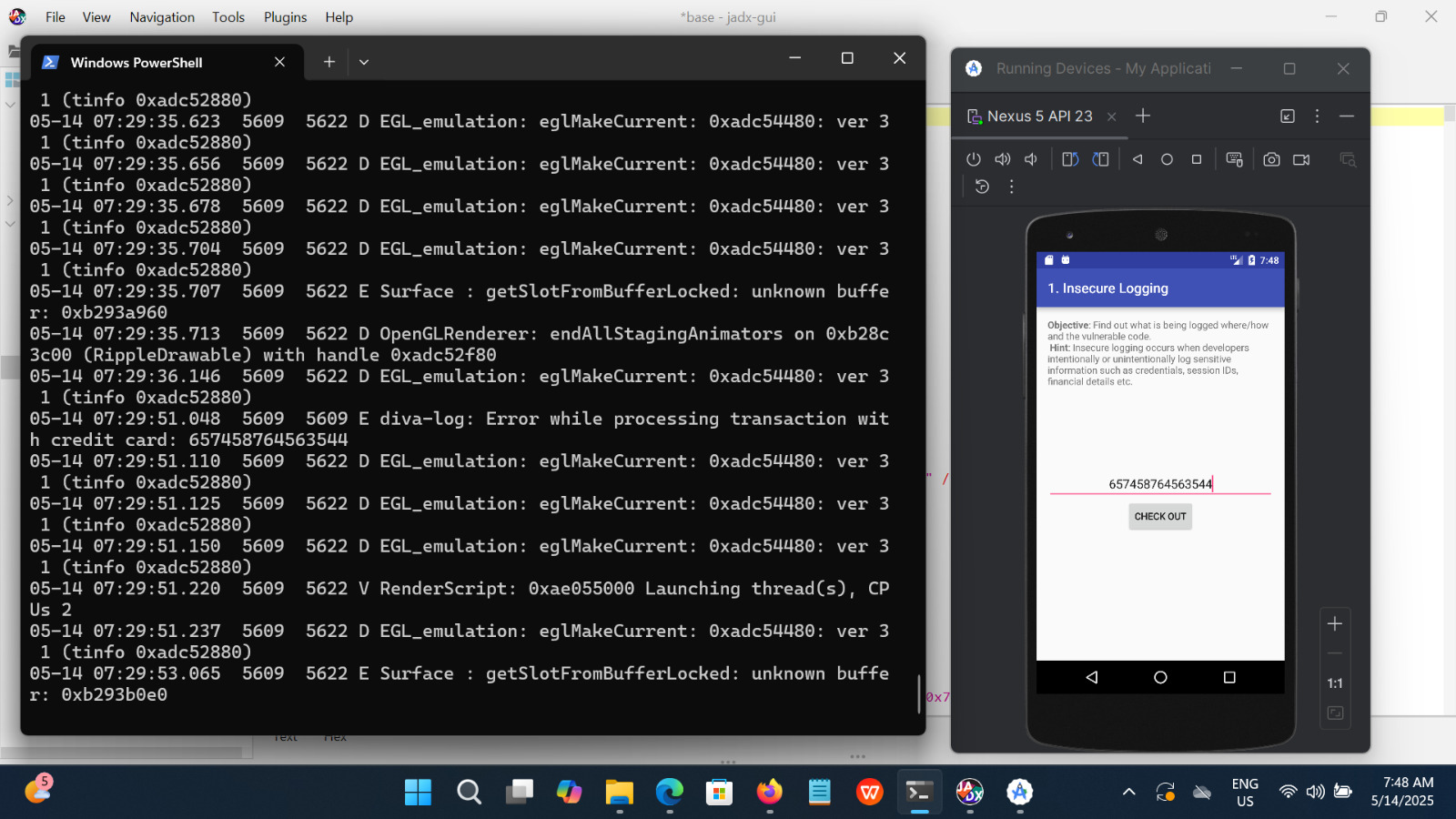


Figure: Logcat showing sensitive data (credit card) logged insecurely.

## 3.2 Insecure Data Storage (Part 1)

Insecure data storage vulnerabilities arise when apps store sensitive information insecurely (e.g., plain text, improperly protected files).  
   
Objective: Test if credentials are stored in shared preferences.  
Method: Input username and password, then inspect shared preference XML file from emulator.  
Result: Credentials were stored in plain text inside shared \_prefs.  
Recommendation: Encrypt sensitive information before storage and apply strict access permissions.

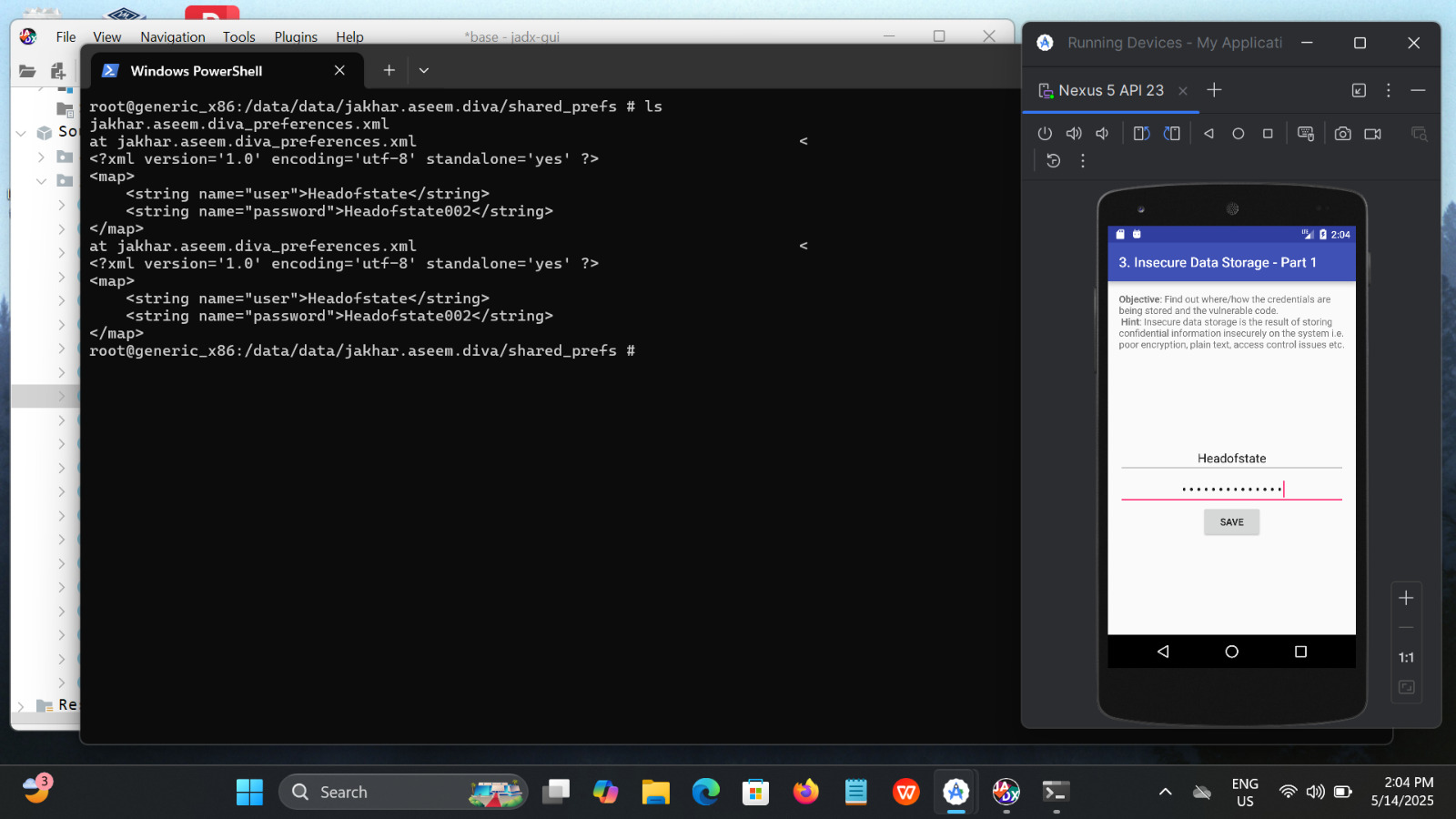


Figure: Shared preferences file storing plaintext username and password.

## 3.3 Input Validation Issues (Part 1)

Improper input validation allows attackers to manipulate application behavior or access unauthorized data.  
  
Objective: Bypass normal user access using a crafted SQL-like input.  
Method: Input `'OR 1=1--` into the input field to retrieve data of all users.  
Result: Application displayed details of all three users including passwords and credit card numbers.  
Recommendation: Sanitize and validate user inputs. Use parameterized queries or ORM frameworks to prevent SQL injection-like flaws.

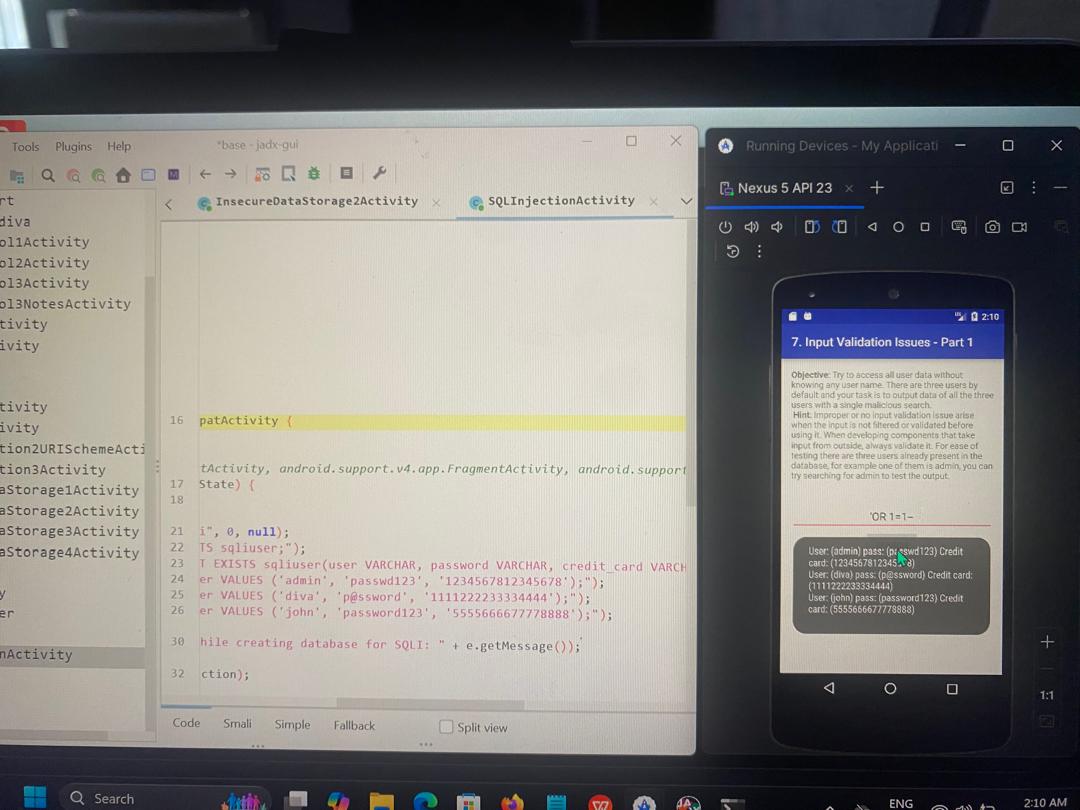


Figure: DVMA interface showing unauthorized access to all user records through invalid input.

# 4. Conclusion

The testing revealed critical security weaknesses in DVMA including insecure logging, data storage, and insufficient input validation. Each vulnerability demonstrates how insecure coding practices can lead to significant risk exposures. Developers must implement secure storage, log hygiene, and rigorous input validation to ensure mobile app security.