COLLEGE OF ENGINEERING

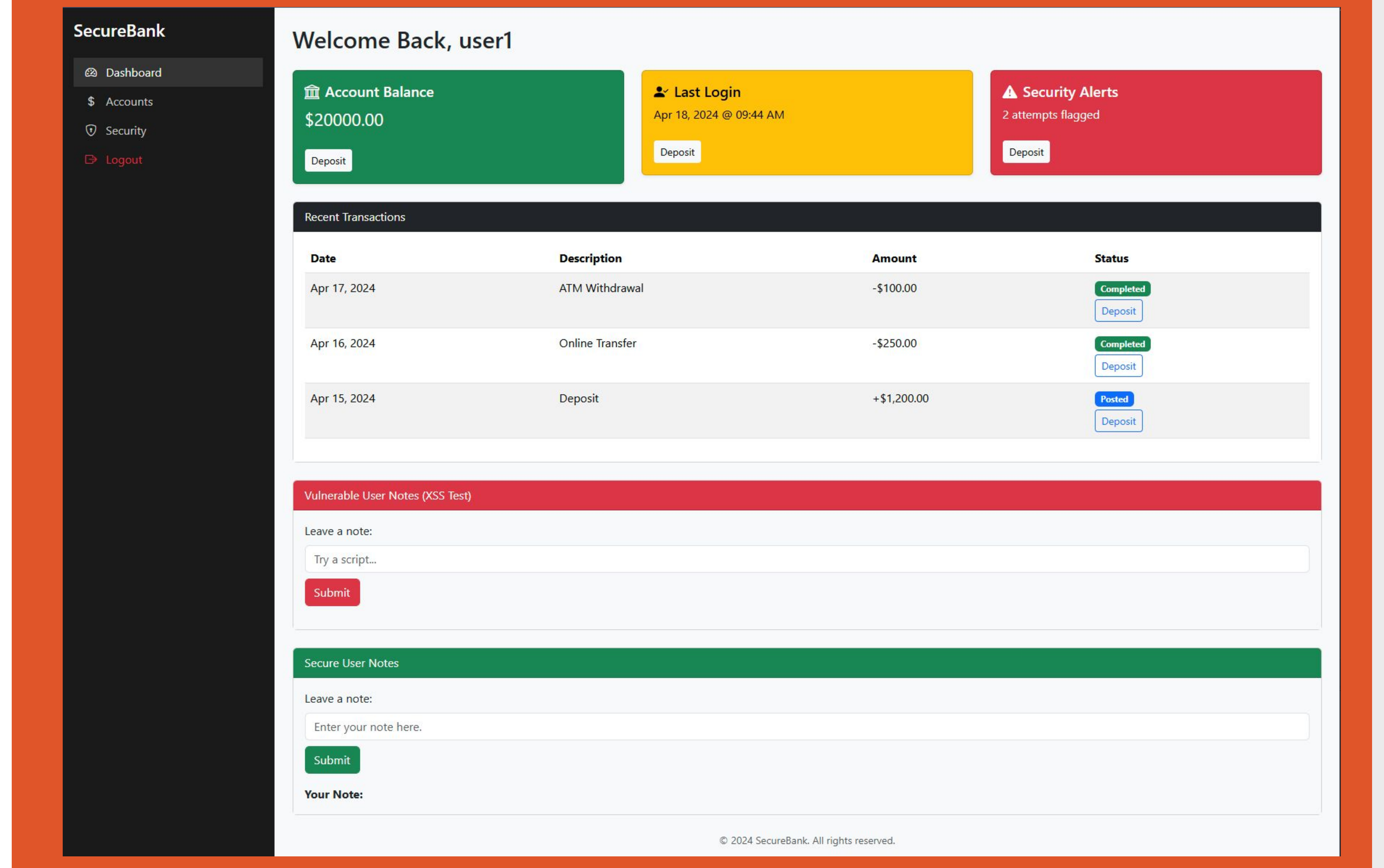
Web Security Research Project

by Manuel Ramirez and Klaus Menendez

Web Security remains a major concern for developers and their users. This project focuses on a few of the most common web security issues. We demonstrate how they are exploited and how we can guard against them.

Project Background and Goals

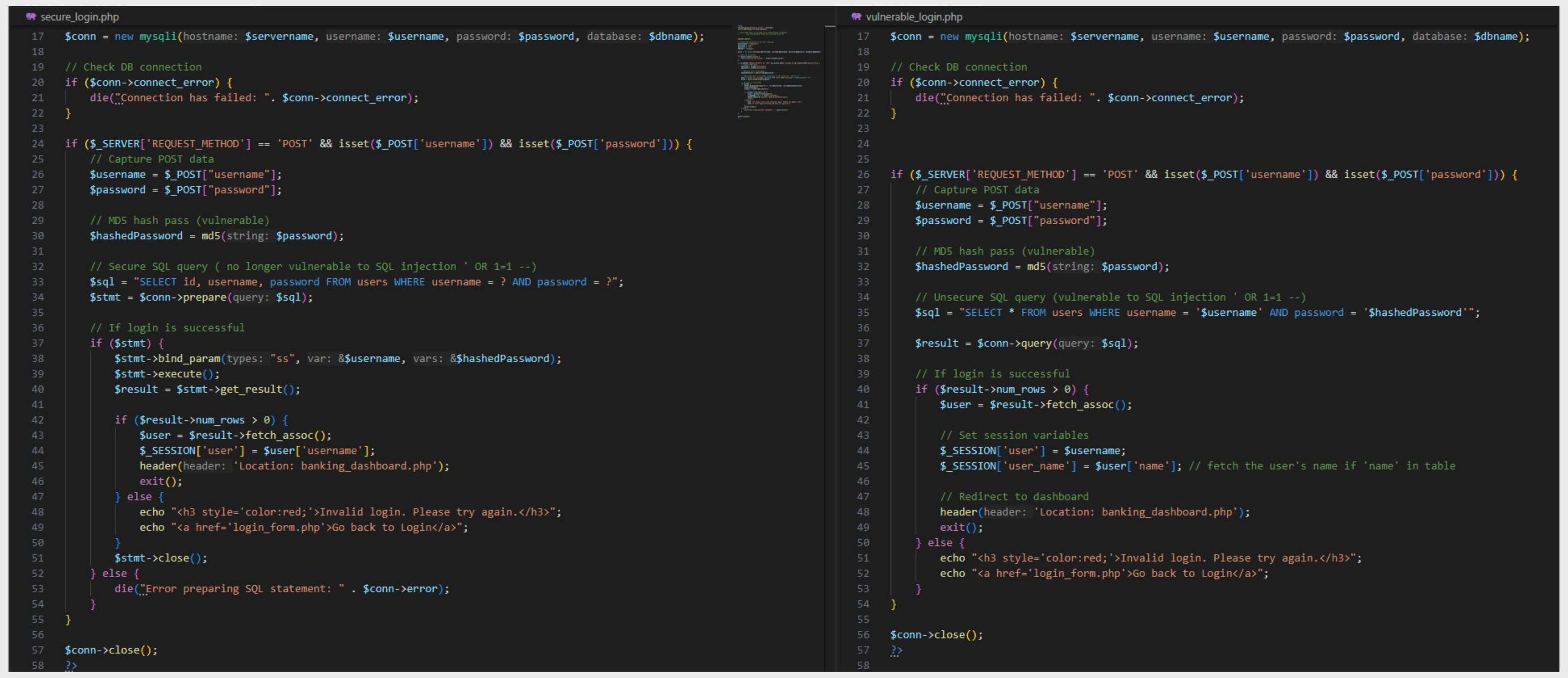
- For our project we used the OWASP Top Ten list to choose from some of today's most common web application security risks.
- Centered around a mock bank dashboard interface we set our to demonstrate how insecure web development practices can lead to serious security vulnerabilities in the real world.
- The application demonstrates exploits such as reflected XSS, SQL injection, insecure password cracking, and file upload vulnerabilities.
- Interactive demonstrations allow users to learn how thse exploits are utilized on insecure code and then how they can be prevented with secure coding practices.



Tech Used

- HTML and Bootstrap for our dashboard UI
- PHP and JavaScript for our backend logic
- SQL & and MariaDB for our password storage
- Apache2 server to host locally
- SQLmap to simulate sql injection on insecure login form
- HashCat to help crack insecure passwords

Oregon State University



Vulnerable vs Secure

Using different versions of the same feature, we implemented vulnerable and secure code examples to use in our database. The above code shows our SQL injection vulnerability which allows users to bypass authentication due to non-parameterized SQL queries.

Users can choose to log in using either the secure or vulnerable methods to see how they work. Rather than creating two applications, one secure and one vulnerable, we decided to include a toggle to make usage more simple.

SQL Injection Vulnerability in the SecureBank Login System

This document details a SQL Injection vulnerability identified in the initial implementation of the SecureBank login system and the subsequent steps taken to patch this critical security flaw.

Vulnerability Description

Vulnerability Type: SQL Injection

Location: `vulnerable_login.php`

Description: The original login script (`vulnerable_login.php`) was susceptible to SQL injection due to the insecure construction of the SQL query used to authenticate users. The script directly embedded user-provided input (the username) into the SQL query string without proper sanitization or the use of parameterized queries.

Vulnerable Code Snippet:

```
$sql = "SELECT * FROM users WHERE username = '$username' AND password = '" . md5($password) . "'";
$result = $conn->query($sql);
```

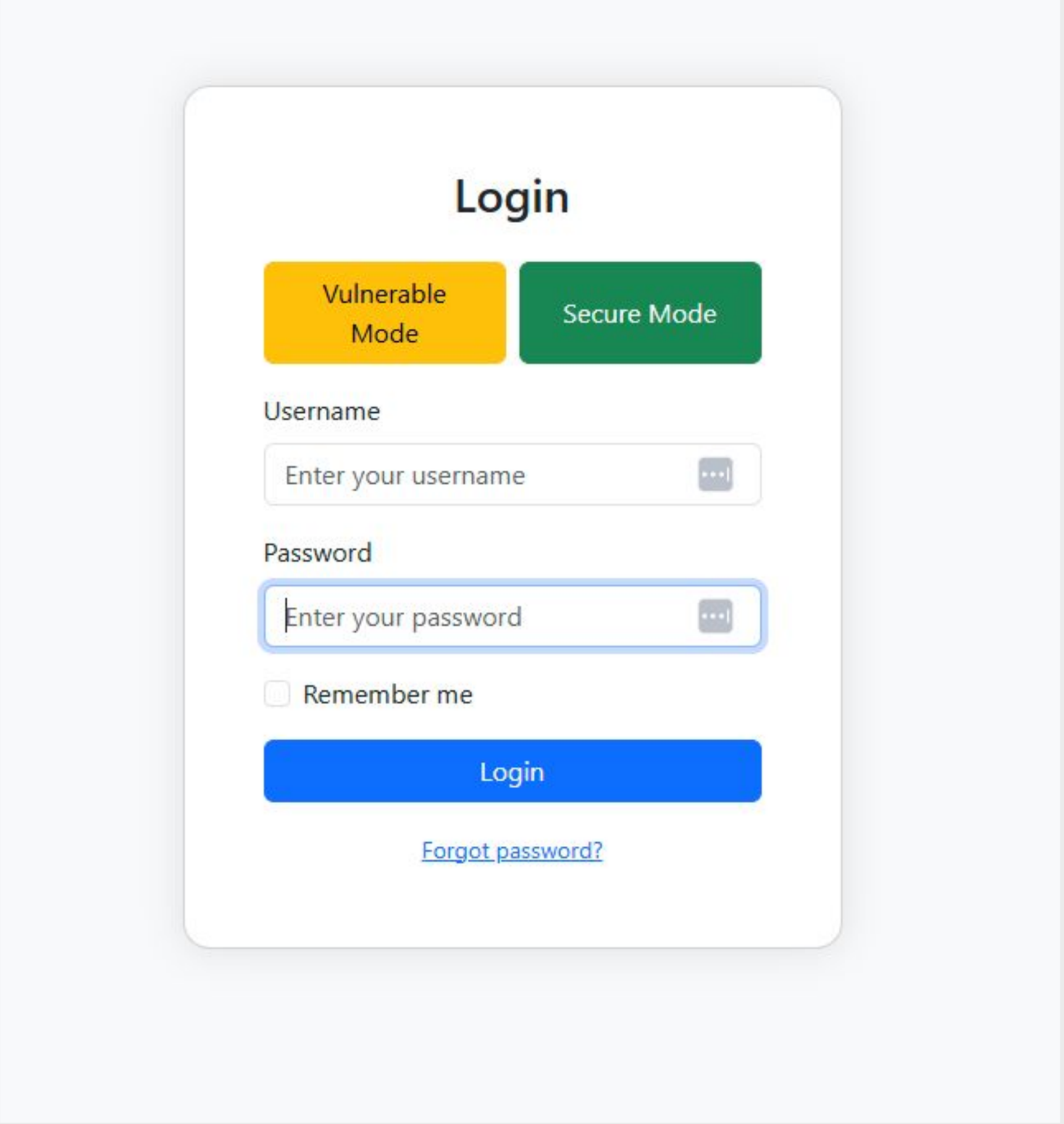
In this vulnerable code, the $username variable, which is directly taken from user input, is concatenated into the SQL query. This allows a malicious user to inject arbitrary SQL code by crafting a specific input for the username field.

Exploitation Scenario: An attacker could exploit this vulnerability by providing a specially crafted username, such as:

```
' OR 1=1 --
```

When this input is embedded into the SQL query, the resulting query becomes:

```
SELECT * FROM users WHERE username = '' OR 1=1 -- ' AND password = '...'
```



Detailed Explanations

Using different versions of the same feature, we implemented vulnerable and secure code examples to use in our database. The above code shows our SQL injection vulnerability which allows users to bypass authentication due to non-parameterized SQL queries.

Users can choose to log in using either the secure or vulnerable methods to see how they work. Rather than creating two applications, one secure and one vulnerable, we decided to include a toggle to make usage more simple.

Bcrypt Hashed Password Crack Attempt

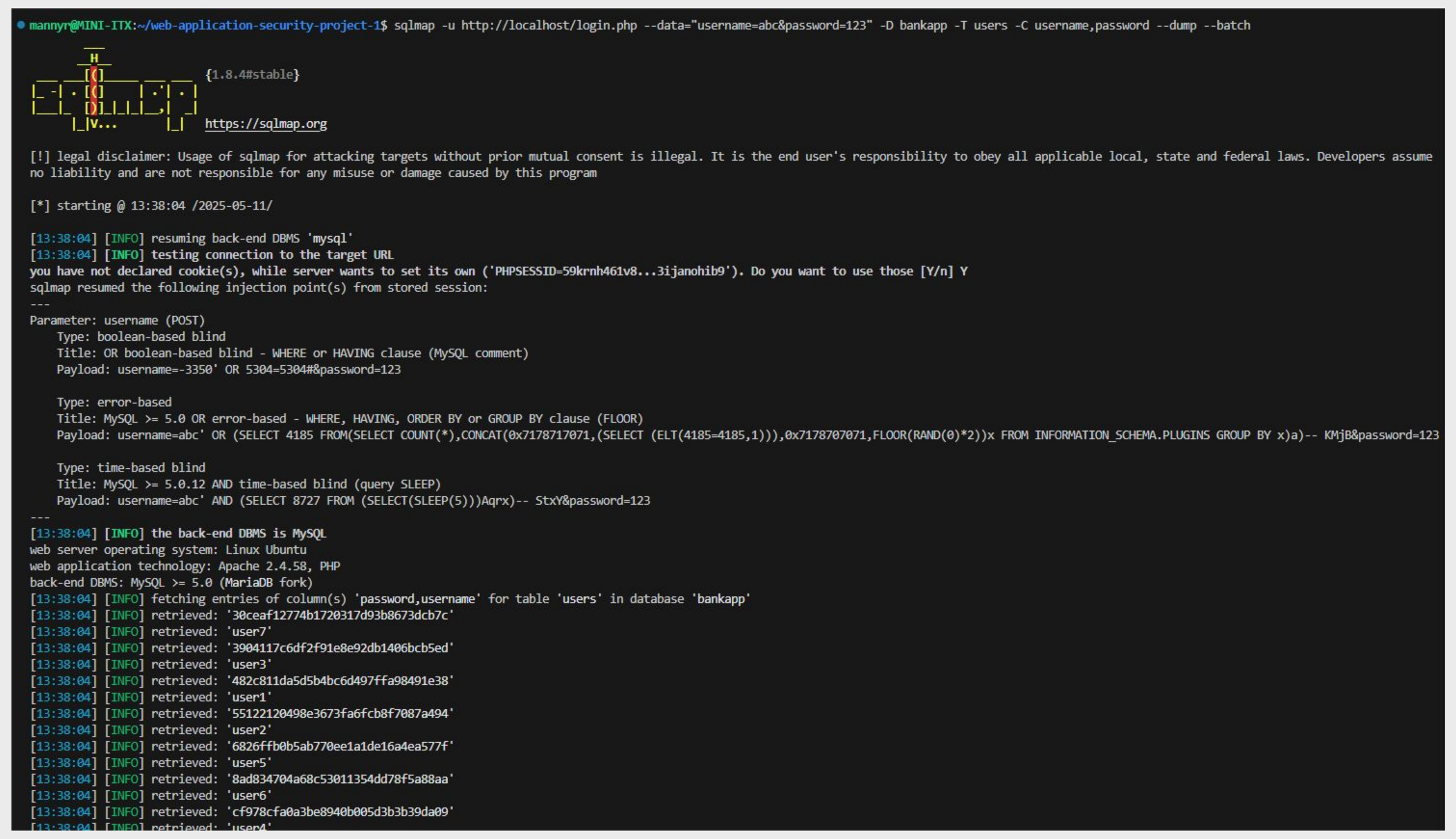
```
Session..........: hashcat
Status...........: Running
Hash.Mode........: 3200 (bcrypt $2*$, Blowfish (Unix))
Hash.Target......: bcryptHashes.txt
Time.Started.....: Sat May 17 17:49:39 2025, (33 secs)
Time.Estimated...: Sun Jun  8 05:14:14 2025, (21 days, 11 hours)
```

MD5 Hashed Passwords

```
MariaDB [bankapp]> SELECT * FROM users;
+----+-------------------+----------------------------------+
| id | email             | password                         |
+----+-------------------+----------------------------------+
|  1 | user1@example.com | 482c811da5d5b4bc6d497ffa98491e38 |
|  2 | user2@example.com | 55122120498e3673fa6fcb8f7087a494 |
|  3 | user3@example.com | 3904117c6df2f91e8e92db1406bcb5ed |
|  4 | user4@example.com | cf978cfa0a3be8940b005d3b3b39da09 |
|  5 | user5@example.com | 6826ffb0b5ab770ee1a1de16a4ea577f |
|  6 | user6@example.com | 8ad834704a68c53011354dd78f5a88aa |
|  7 | user7@example.com | 30ceaf12774b1720317d93b8673dcb7c |
+----+-------------------+----------------------------------+
```

- Exploiting multiple vulnerabilities, we were able to use SQL injection to steal our stored passwords and then use HashCat to crack the unsalted MD5 hashes almost instantly.
- Cracking Bcrypt hashes would have taken 21 days

SQLmap Stealing Passwords



Learning Outcomes

- We learned a lot about the importance of secure development practices and the negative impacts of ignoring them.
- Implementing a secure website is almost as hard as intentionally making it insecure.
- Vulnerabilities aren't always easy to identify and patch, some may be hidden and eve go undetected for a long time until they lead to data breaches or worse.
- Vulnerabilities vary in degree of danger, some of the most dangerous ones can be guarded against by simply informing oneself and using good coding practices.
- There are many great resources for web developers to stay informed on the latest security vulnerabilities being exploited and how they can be guarded against. Our greatest resource: https://owasp.org/www-project-top-ten/