Final Technical Report: Secure Coding Project

Students:
Daniel Firley, 312124498
Yuval Geta, 211562889
Maor Sheiman, 322659780
Ilay Weizman, 318760725
Yakira Siboni, 208499426
Daniel Chani, 322757840

1. Project Overview

This project demonstrates an enterprise information system ("Communication_LTD") featuring user and client management. The system is deployed in two distinct versions to illustrate the contrast between vulnerable and secure code:

1. Vulnerable Version (app_vulnerable.py): Intentionally contains critical security flaws (OWASP Top 10) allowing Authentication Bypass, Data Exfiltration, and Code Execution.
2. Secure Version (app_secure.py): Implements "Defensive Programming" principles to mitigate these attacks effectively.

The system is built on Python Flask utilizing a MySQL database (XAMPP environment).

2. Attack Surface Analysis

A. Authentication Bypass via SQL Injection (UNION-Based Imposter)

Despite implementing strong password hashing (HMAC-SHA256), the system is vulnerable to SQL Injection due to unsafe string concatenation in the login query.

- The Vulnerability: The query SELECT * FROM users WHERE username = '{user_input}' allows attackers to manipulate the SQL structure.
- The Attack Vector (Imposter Attack): Since the server verifies passwords programmatically, a simple SQL bypass (e.g., ' OR 1=1) is insufficient. The attacker must inject a forged identity:
    1. The attacker generates a valid Hash/Salt pair for a known password (e.g., password123) using the external script hacker_tool.py.
    2. Using the UNION operator, the attacker injects a row containing this valid Hash/Salt but sets the username column to the target identity (e.g., admin).
    3. The server fetches the injected row and validates the input password against the injected hash. Since they match, the server authorizes the login.

B. Data Exfiltration

Leveraging the same UNION vulnerability, the attacker can extract sensitive data and display it on the main Dashboard:

- Fingerprinting: Injecting @@version into the username column reveals the database version.

- Data Dump: Using the GROUP_CONCAT function allows dumping the entire username/password database into a single string displayed in the "Welcome" message.

C. Cross-Site Scripting (XSS)

- Stored XSS: The application uses the |safe filter in HTML templates, instructing the browser to execute raw scripts injected into the Description field.
- Malicious Links: Lack of server-side validation on URL fields allows injecting javascript:alert(1) payloads.

3. Defense Implementation (Security Controls)

The secure version (app_secure.py) mitigates all the above threats through infrastructure hardening:

A. Preventing SQL Injection - Parameterized Queries

Instead of concatenation, the code uses Placeholders (%s).

- Mechanism: The MySQL driver sends the query structure and the data separately to the database engine.
- Result: Even if the attacker inputs ' UNION SELECT..., the database treats it as a literal string value and searches for a user with that specific name. The malicious command is never executed.

B. Preventing XSS - Context-Aware Encoding

- The |safe filter was removed. The Jinja2 engine performs Auto-Escaping by default.
- Dangerous characters like < are converted to HTML entities (<), rendering them as harmless text instead of executable code.

C. Input Validation

- Server-side Regex validation was implemented for URL fields.
- The server strictly enforces that URLs must start with http:// or https://, blocking javascript: vectors before they reach the database.

4. Execution & Demo Guide

1. Setup: Ensure XAMPP is running and execute python db_setup.py.
2. Payload Generation: Run python hacker_tool.py and copy the desired payload (e.g., Payload [1] for Admin Login).
3. The Attack (Vulnerable):
    - Run app_vulnerable.py.
    - Paste the payload into the Username field and enter password password123.
    - Result: Successful login as Admin.
4. The Defense (Secure):
    - Run app_secure.py.
    - Attempt the exact same attack.
    - Result: "Invalid Credentials" error – The attack is successfully blocked.