Building to Break A "Vulnerable by Design" Approach to Web Security

COMP6841

Security Engineering

Project Report

Author Details

- **Name** - Sagar
- **Course** - COMP6841
- **Tutorial Group** – H11B

Results

The primary result of this project is a fully functional, locally hostable "Vulnerable by Design" web application. This application, a "Student-Tutor Forum," was built from the ground up using Python and Flask to serve as a tangible sandbox for demonstrating critical web security concepts. The project successfully implements and provides proof-of-concept exploits for five distinct, high-impact vulnerabilities from the OWASP Top 10 2021 list:

1. **A01:2021 - Broken Access Control** (including Insecure Direct Object Reference (IDOR))
2. **A03:2021 - Injection** (SQL Injection and Stored Cross-Site Scripting)
3. **A10:2021 - Server-Side Request Forgery (SSRF)**

The technical depth required for COMP6841 is demonstrated through the complexity of these exploits, which go beyond simple identification. For instance, the SQL Injection vulnerabilities allow for login bypass, sensitive data exfiltration, and full account privilege escalation. The SSRF exploit successfully bypasses a simulated firewall to exfiltrate data from a protected internal endpoint. These practical demonstrations showcase a mastery of the chosen security topics.

In addition to the vulnerable application itself, the project deliverables include a secure version of the codebase with all flaws remediated, a detailed final report documenting the entire project, and a 3-minute video presentation. The video provides a clear, engaging, and concise demonstration of the key exploits whilst communicating the project's outcomes and the severity of each vulnerability.

What I did

The project was executed over eight weeks, following a structured, iterative approach.

- **Weeks 1-3 (Foundation & Research):** The initial phase involved project ideation, researching the OWASP Top 10 to select impactful vulnerabilities, and planning the application's architecture. My initial goal was to implement 2-3 vulnerabilities, but this research sparked an interest that led me to expand the scope to five, which eventually increased the project’s complexity.

- **Weeks 4-7 (Development & Implementation):** I dedicated this period to building the application from scratch. This involved setting up the Flask backend, designing the SQLite database schema, and coding the frontend. I started off by building a core functional feature (e.g., login, posting, user profiles) and then deliberately implementing the associated vulnerability. For each flaw, I developed a clear proof-of-concept exploit, such as the privilege escalation payload for SQLi or the cookie-stealing script for XSS. This required a constant mental shift between a "builder" and "attacker" mindset.
- **Week 8 (Finalisation):** The final week was focused on documentation and presentation. I assembled the detailed final report, which includes in-depth analysis and remediation steps for each vulnerability. I also wrote the script for, recorded, and edited the 3-minute video demonstration, ensuring it clearly communicated the project's narrative. A significant challenge during this phase was time management. The initial plan to deploy the app to a cloud service was abandoned to ensure the core deliverables (the application and its documentation) were completed to the highest standard.

How I was challenged

This project was a significant challenge that pushed me far beyond my comfort zone as a developer and fostered substantial personal growth.

The greatest challenge was learning to **deliberately write insecure yet functional code**. As a developer, my instincts are geared towards best practices and security. Actively fighting those instincts to create a specific vulnerability required a deep, first-principles understanding of why a piece of code is insecure. It forced me to move from simply using a secure framework to understanding the mechanics of the attacks it prevents.

Another major challenge was **developing realistic and compelling exploits**. It wasn't enough to just have a vulnerability. I needed to show its impact. The SSRF demonstration was a prime example of this struggle. My initial thought was to demonstrate reading a local file from the server something that would give away some credentials, but I quickly realised this had a logical flaw for a live demonstration - how would an attacker know the exact filename to request? Or even if he would, how would he get the location and given the code is a localhost, how does he even get the file This wouldn't be a convincing exploit. After several failed attempts to create a realistic scenario, I ultimately re-designed the vulnerability itself. I created a "hidden" internal API endpoint that was protected from direct access. The exploit was then to trick the server into accessing this page via the profile picture URL. The application's flawed error handling would then leak the contents of this forbidden page, including administrator credentials, back to the attacker. Overcoming this setback was a critical learning experience. It taught me not only about problem-solving but also about how multiple small issues like improper URL validation and overly verbose error messages combine to create a critical vulnerability.

Finally, managing the **project's scope** was a personal challenge. My enthusiasm for the subject led me to expand the project from three vulnerabilities to five. While this made the final output more impressive, it also put immense pressure on my time management. This experience taught me a valuable lesson about the importance of rigorous planning and the discipline required to adhere to a predefined scope, even when new, exciting ideas emerge. Ultimately, this project transformed my perspective; I am no longer just a developer who uses security features, but an engineer who understands why they are necessary.