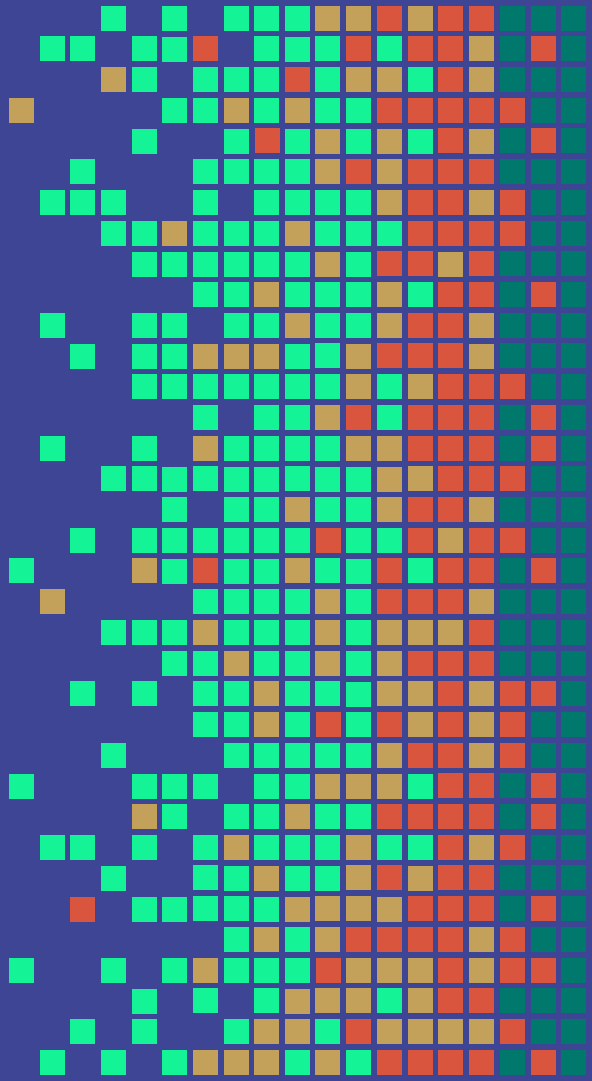
SecureDove

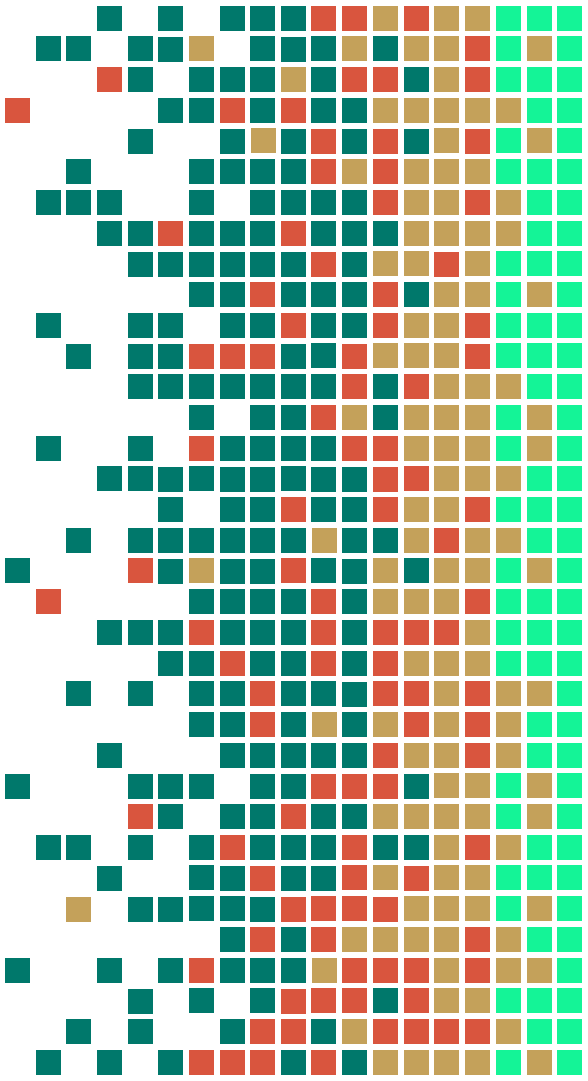
Team: DAAP-SccDov-518
Members: Aryun Kumur, Durshil Juyuni, Alcjundro MilIcr-Gonzulcz, Purtho Bhuttuchuryu
Presentation Link:
https://drive.google.com/file/d/1qv5DOccbKHHR16XnEfmSd4no5IfX5_P8/view?usp=sharing



Introductions

- Aryan Kumar - Contact Person, setting up barebones application, cracking the password, session hijacking, HTTPS, JS disabled.
- Darshil Jayani - Profile management, file upload, file sharing, Wireshark Analysis, XSS and SQL Injection, Findings summary writing
- Alejandro Miller-Gonzalez - Email integration, password reset, hashing passwords, securing database
- Partho Bhattacharya - Initial template building, groups functionality in the app, recaptcha, side channel, rate limiting

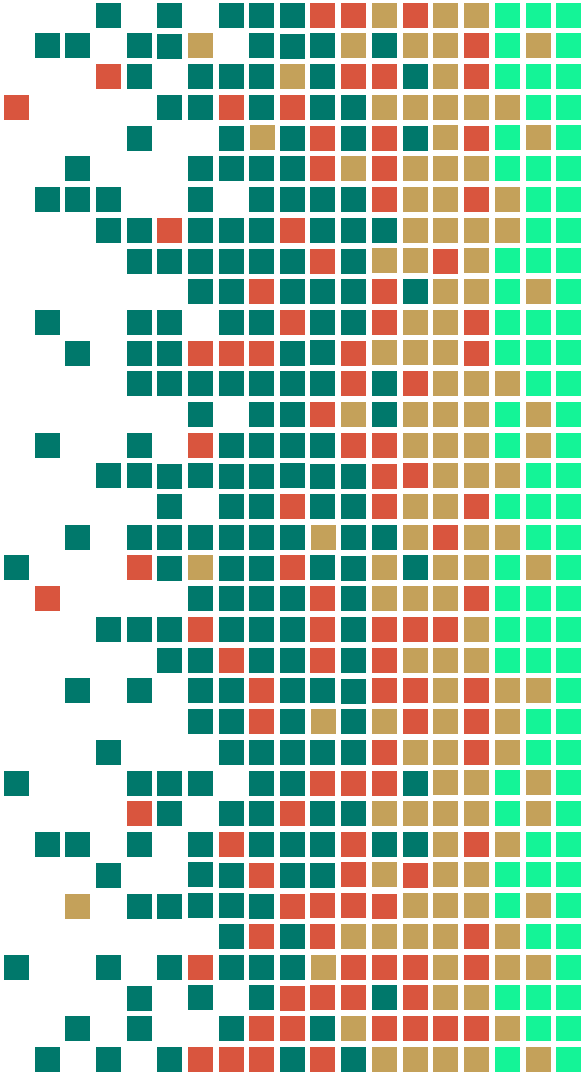
1. Dclivcrublc Onc



Rccruiting

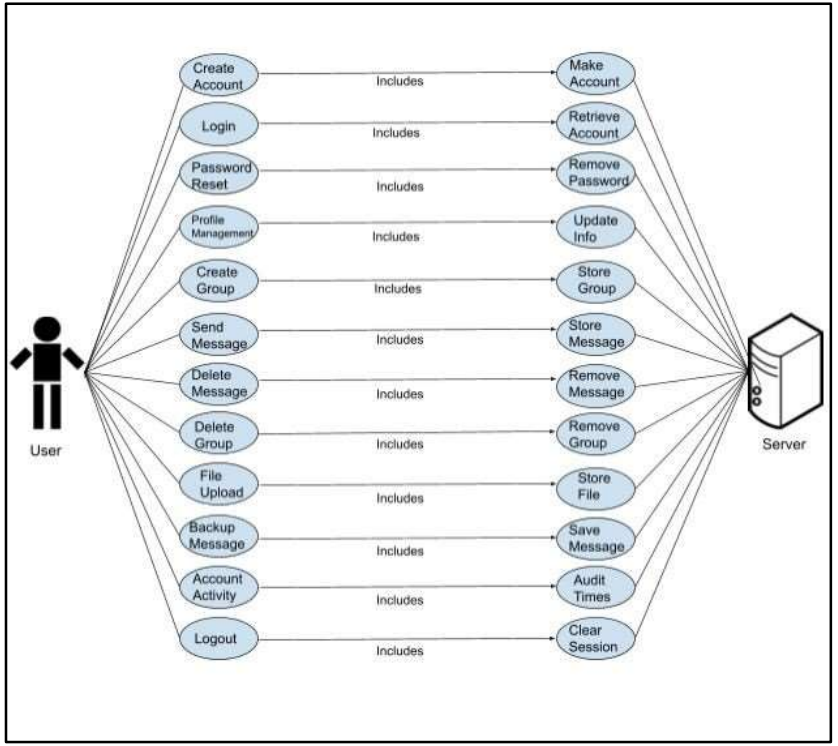
- We put out a post on Piazza and got 4 people together to start working.
- Together we came up on the idea of having Secure Dove be the project topic we will work on.
- We brainstormed ideas about topics and how in-depth we wanna go.

2. Dclivcrublc Two



2.1 Rcquircmcnts

- We created a list of design requirements that we thought would work best with our project.
- We completed this using use case modeling, developing the UML Use Case Diagram and accompanying use case tables that detail every interaction between the user, server, and database.
- We decided to build an application that lets users securely log in and send and delete encrypted messages.

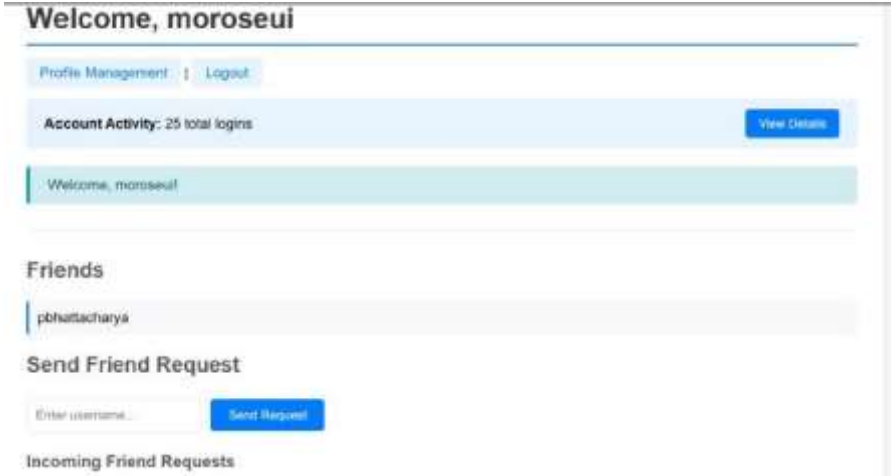


2.1 Sccurity Plun

- To plan the security of our application, we broke down the high-level goals of the CIA triad into specific security requirements for SecureDove. Each requirement was paired with concrete methods for achieving it like using encryption to ensure message confidentiality.
- To evaluate the application's security, we created a percentage-based scoring system that measured compliance with these requirements. The target score was 100%, and our goal was to maintain this level consistently.

2.2 Dcsign

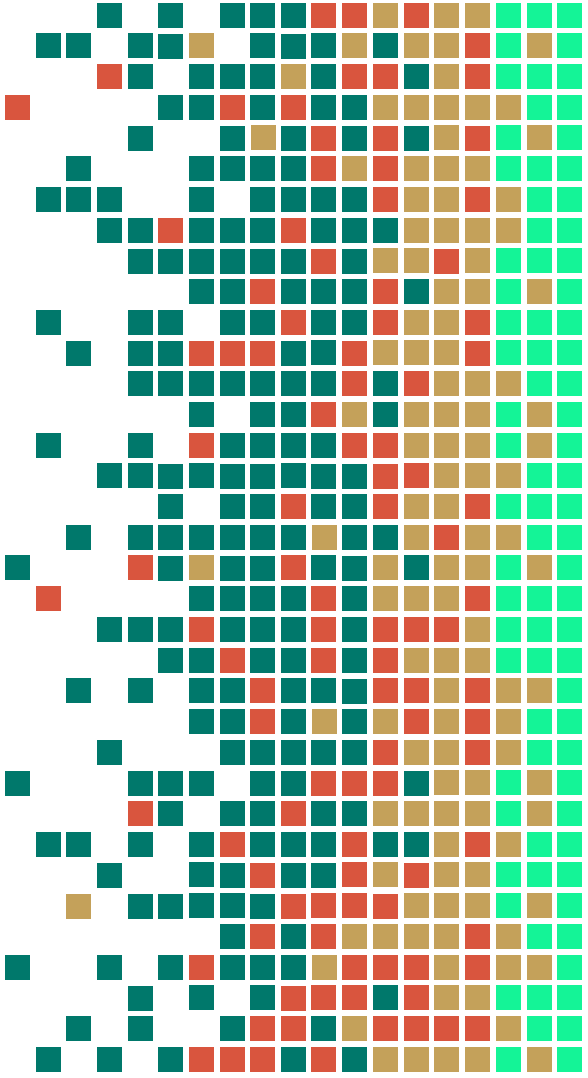
- Password Reset Feature
- Account Password Management
- HTTPS secured communication
- Individual Messaging
- Account and Message Deletion
- IP Blocking
- Group Creation and Management
- Log Auditing
- Logins Tracked



Expcctutions bc√orc Sturting

- Collaboration with people.
- Communicating with team
- Planning everything out with team and constantly updating
- Managing time

3. Dclivcrublc Thrcc

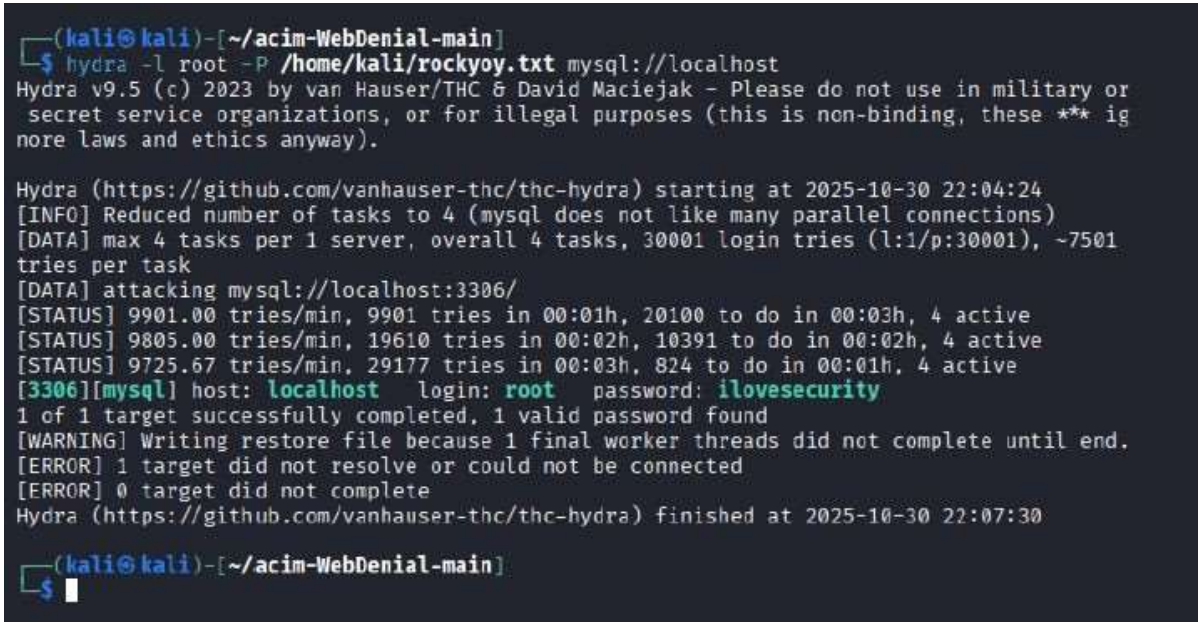


Thc Attucks

- Password Cracking(Dictionary Attack)
- Packet Sniffing
- Session Hijacking
- Denial of Service (DoS)
- Cross Site Scripting (XSS)
- Side Channel

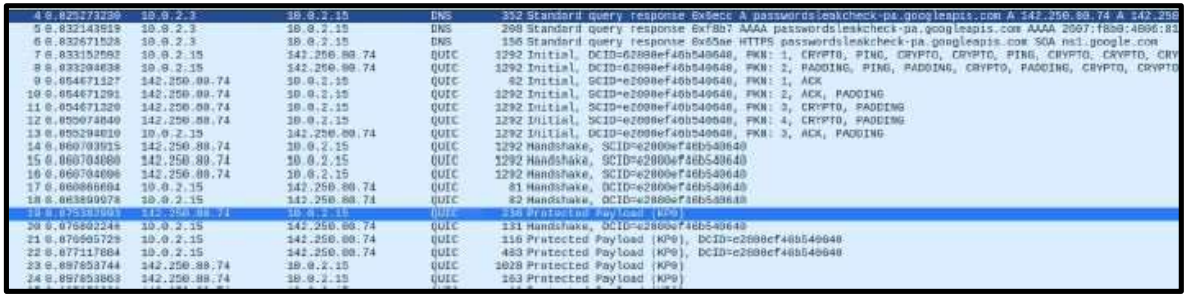
Pussword Crucking

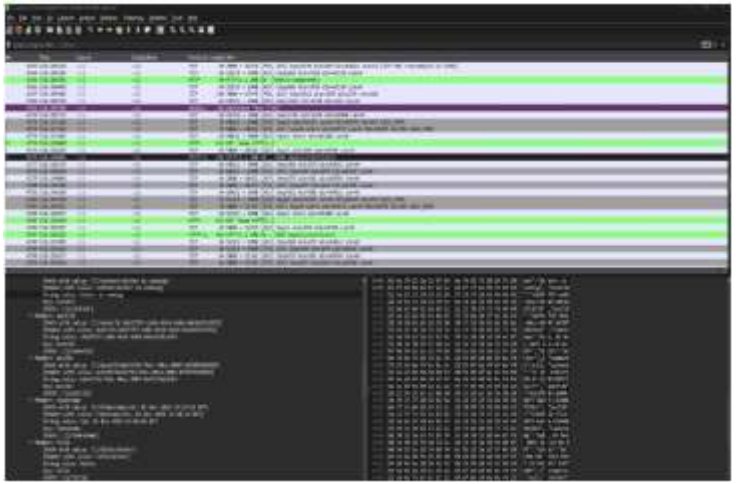
Using basic Nmap scanning, we discovered that the MySQL port was exposed. We then used the commonly known “rockyou.txt” wordlist to perform a dictionary attack on the database credentials and we were able to successfully crack the password.



Puckct Sni√fing

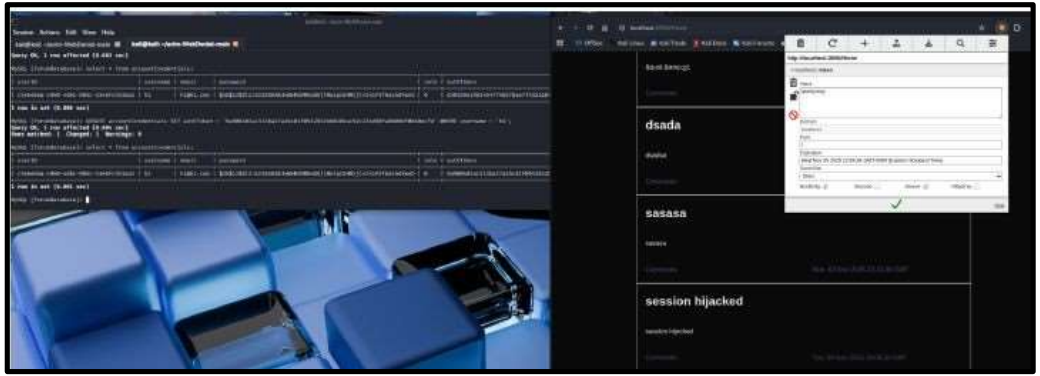
We also performed packet sniffing on the application using Wireshark. While the standard handshakes were encrypted through the QUIC protocol, we found that when a user created a post, the transmitted data could still be captured in plaintext.





Scssion Hijucking

We were also able to perform session hijacking as a result of cracking the MySQL password. By modifying the session cookie stored in the database and syncing it with the cookie in our browser, we were able to impersonate the targeted user.

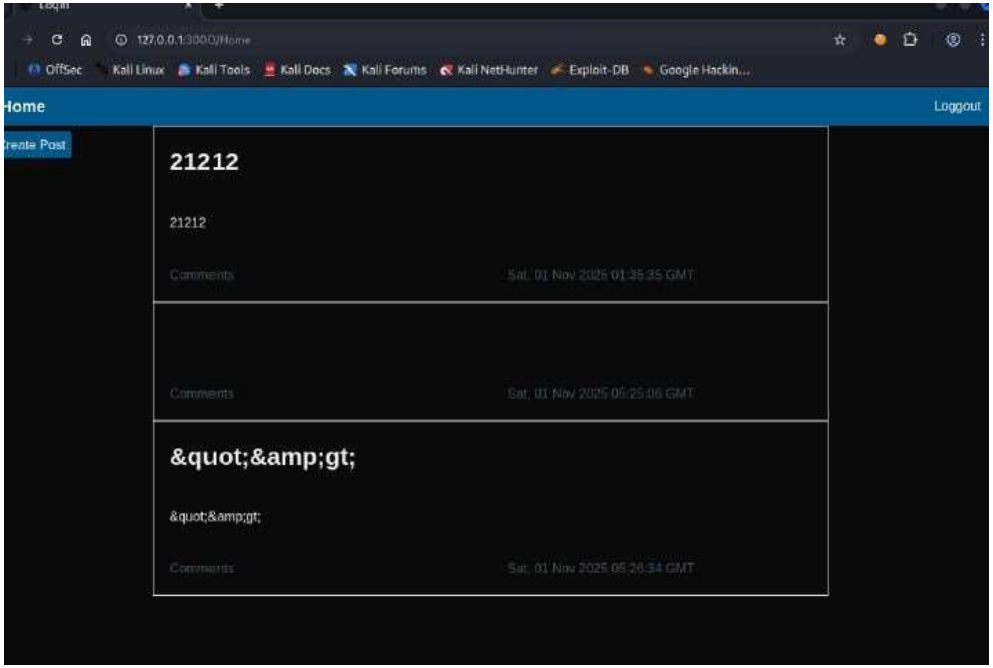


Dcniul O√ Scrvicc

The application relied on the X-Real-IP header for user identification, but this header is fully user-controlled. Although rate limiting exists in __init__.py, it becomes ineffective because attackers can simply rotate the header value to bypass it. Our proof-of-concept shows that brute-force attempts exceed 20 requests in 10 seconds without triggering a 429 error.
We also observed that the gatekeepObj function, which appears designed to ban users, is never called, preventing any 403 Forbidden responses from ever being issued.

Cross Sitc Scripting (XSS)

We checked for XSS by attempting to inject JavaScript into the post feature. The application correctly filtered the input, and the injected code was sanitized rather than executed, confirming that XSS attacks were not possible in this area.



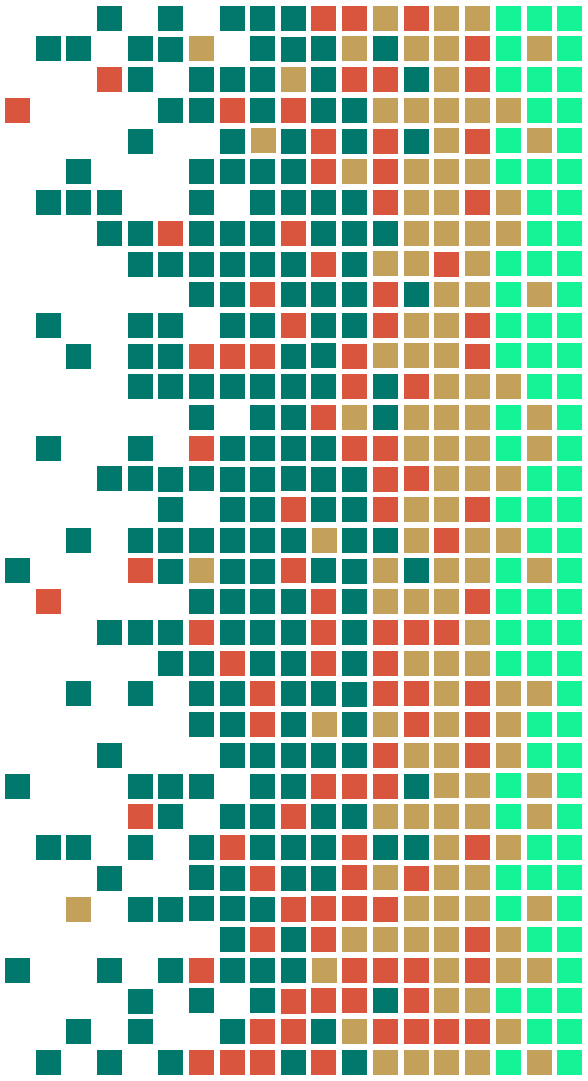
Sidc Chunncl

On sign-up, passwords are hashed with bcrypt. During login, the application re-hashes the provided password and compares it only if the username exists. If the username is invalid, it returns an error immediately. Since bcrypt is slow by design, this creates a measurable timing difference between valid and invalid username checks. As seen in the images we see we can bruteforce existing usernames using the timing difference as leverage.

Expcricncc guincd und Lcssons Lcurncd

- Every software can be broken
- Discovered places to look for common exploits
- Learned to execute basic exploits
- OSINT and Info gathering on the software
- Your team may find something you didnt

4. Dclivcrublc Four

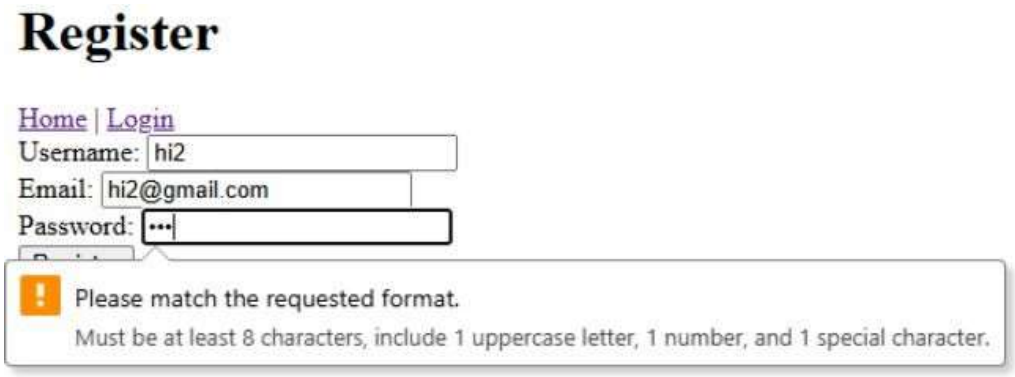


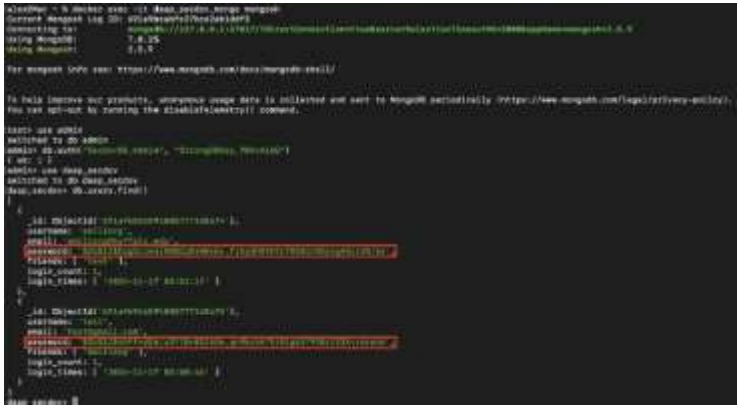
Fixing vulncrubilitics

- Revisiting CIA.
- Hashing store passwords and added password complexity requirements.
- Using HTTPS to prevent sniffing and packet capture.
- Implementing DDOS protection by not letting IPs send random requests.
- Disabling JavaScript so frontend cookies can’t be changed or discovered.
- Sanitizing Input so executable code will not be run.

Pussword Crucking

- Password complexity was added of at least 8 characters with one capital letter, one number and one special character to make it hard to crack leaked passwords.
- Passwords were also hashed when being stored in the database so even if leaks occur they are safe to an extent



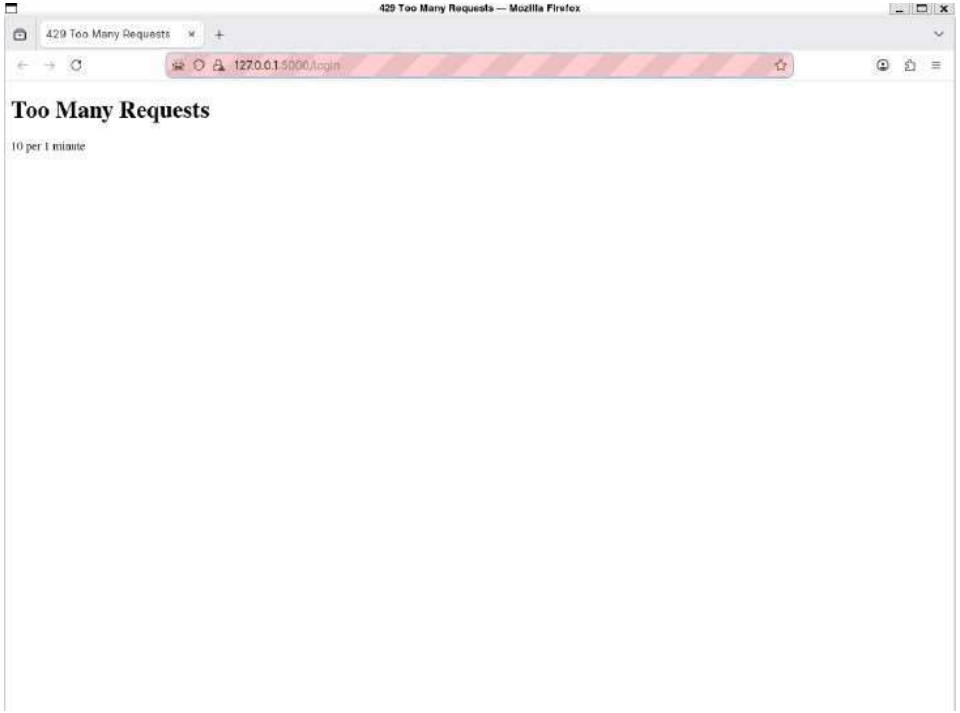


Puckct Sni√fing

- After enabling HTTPS, packet contents became encrypted, preventing attackers from viewing the cookies and message data that were previously exposed in plaintext.

Dcniul o√ Scrvicc(DOS)

- With rate limiting in place, once a user exceeds the allowed number of requests, the server denies additional attempts and responds with a 429 error.

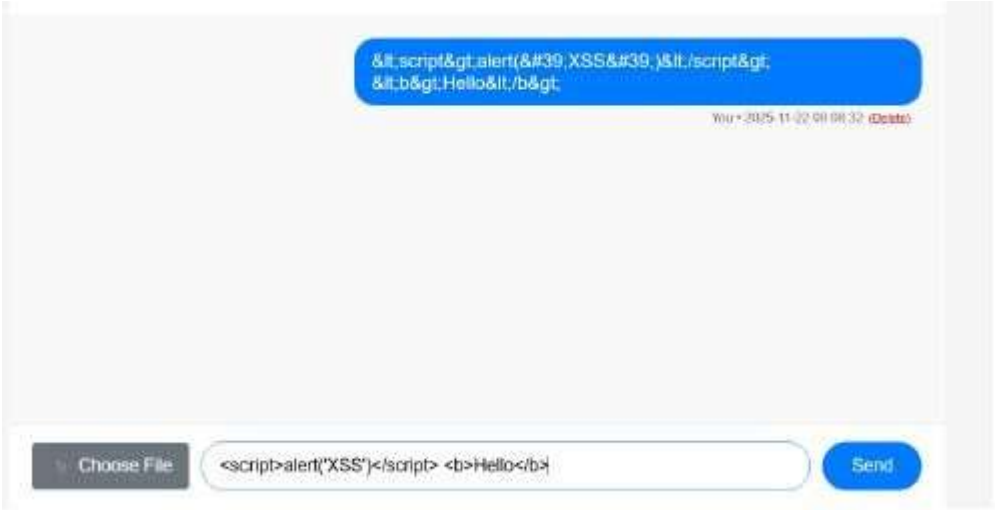


Scssion Hijucking

- With HTTPS, session cookies are no longer visible in sniffed traffic. We further strengthened security by disabling JavaScript execution, eliminating the ability to read or modify cookies from the frontend.

Cross-Sitc Scripting(XSS)

- We sanitized user input removing the dangerous parts of XSS payloads, ensuring the injected content is processed as text rather than executable code.



Ovcrull Lcurning

- Learned how to test for vulnerabilities in a software
- There will always be security issues that will need fixing
- You can’t make something too secure or it might result in availability issues.
- You have to be very adaptable based on the application you are testing and the user base
- You won’t be able to find everything on your own that’s why you work with peers and in a team

THANK YOU!