

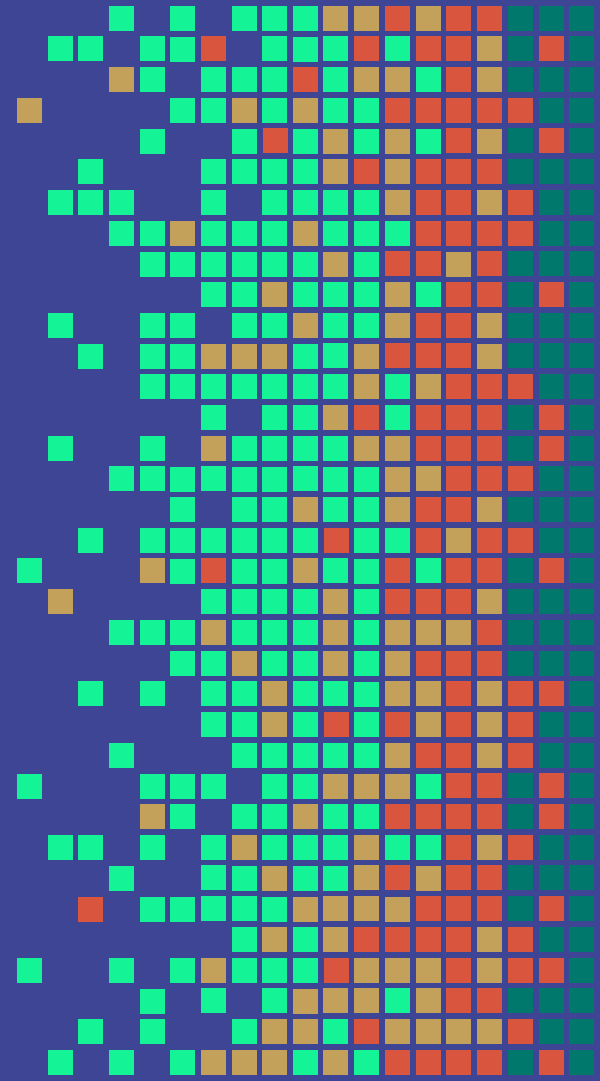
# SecureDove

**Team:** DAAP-SccDov-518

**Members:** Aryun Kumur, Durshil Juyuni, Alcjundro  
Miller-Gonzulcz, Purtho Bhuttachuryu

**Presentation Link:**

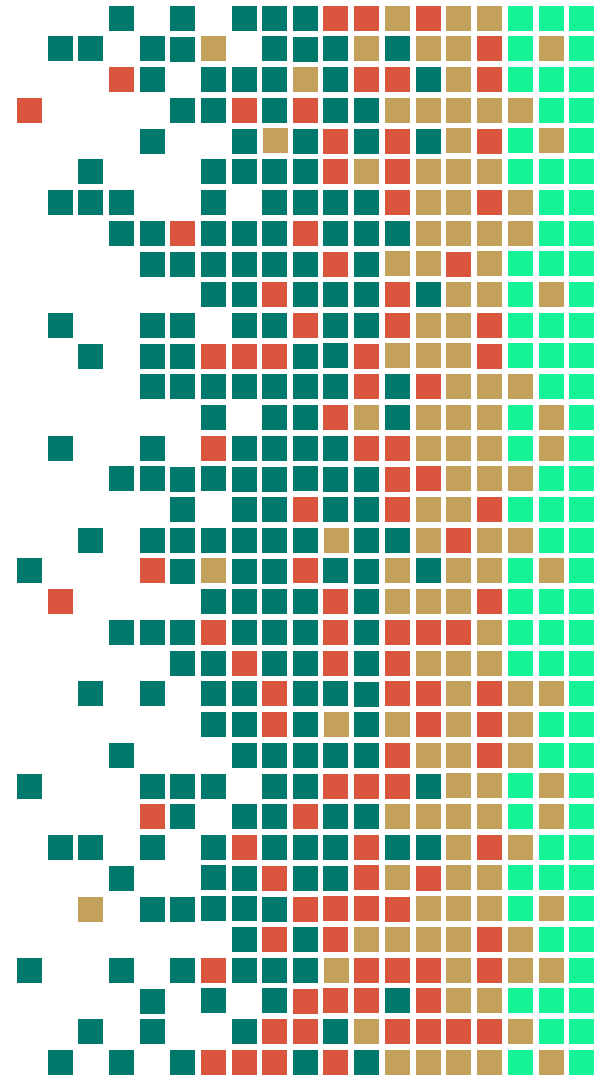
[https://drive.google.com/file/d/1qv5DOocbKHHR16XnEf\\_mSd4no5lfX5\\_P8/view?usp=sharing](https://drive.google.com/file/d/1qv5DOocbKHHR16XnEf_mSd4no5lfX5_P8/view?usp=sharing)



# 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

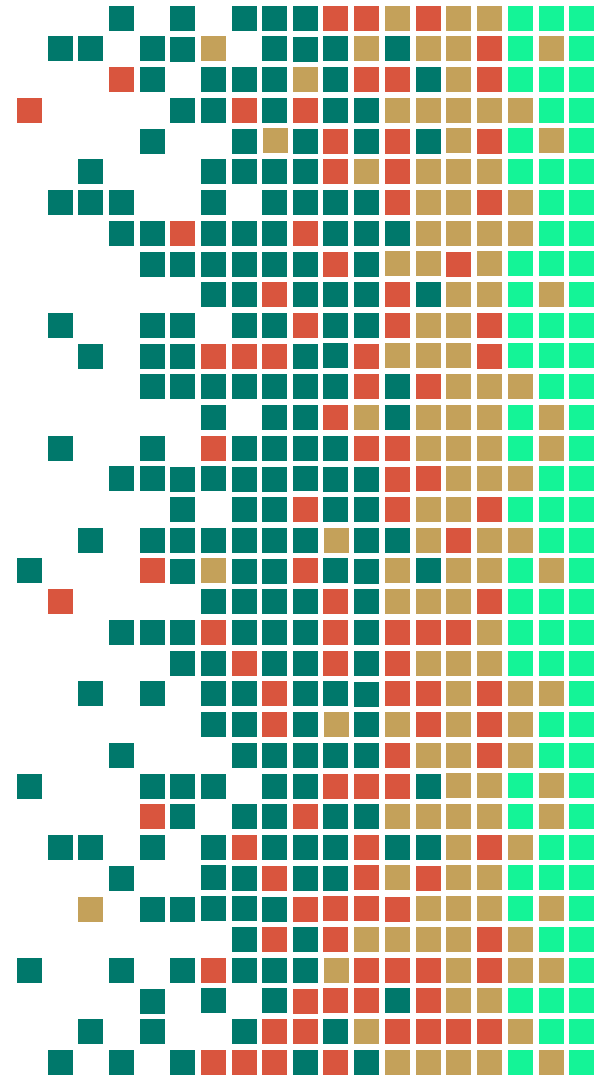
# 1. Dclivcrublc Onc



# Rccruiting

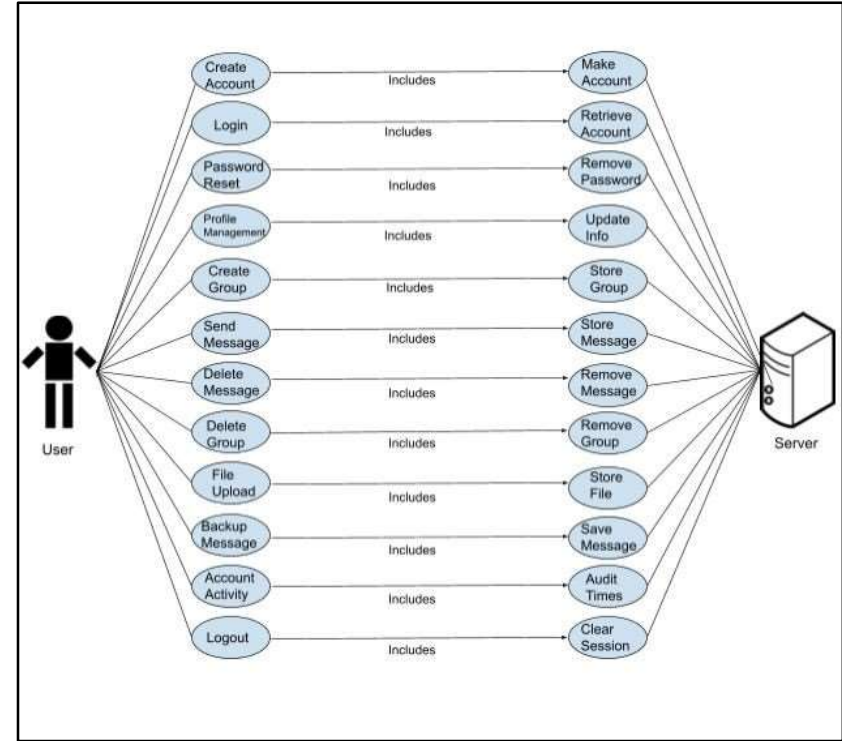
- We put out a post on Piazza and got 4 people together to start working.
- Together we came up on the idea of having Secure Dove be the project topic we will work on.
- We brainstormed ideas about topics and how in-depth we wanna go.

## 2. Deliverable Two



## 2.1 Rcquirments

- 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.



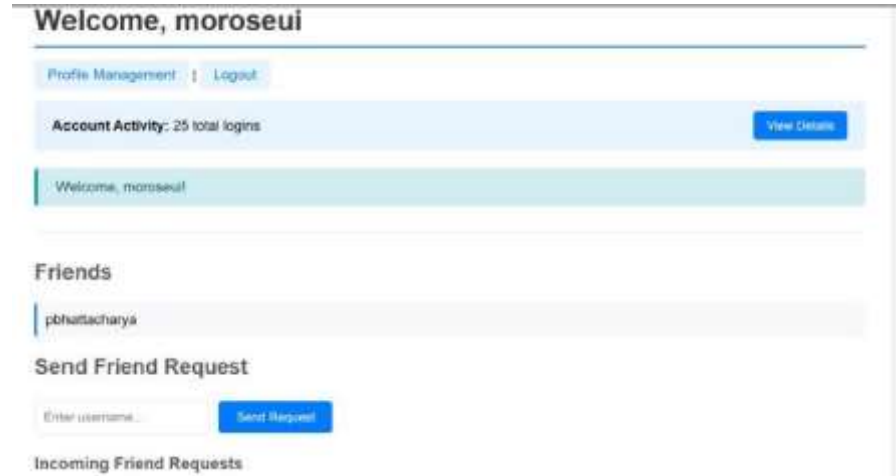
## 2.1 Ssecurity Plun

- 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

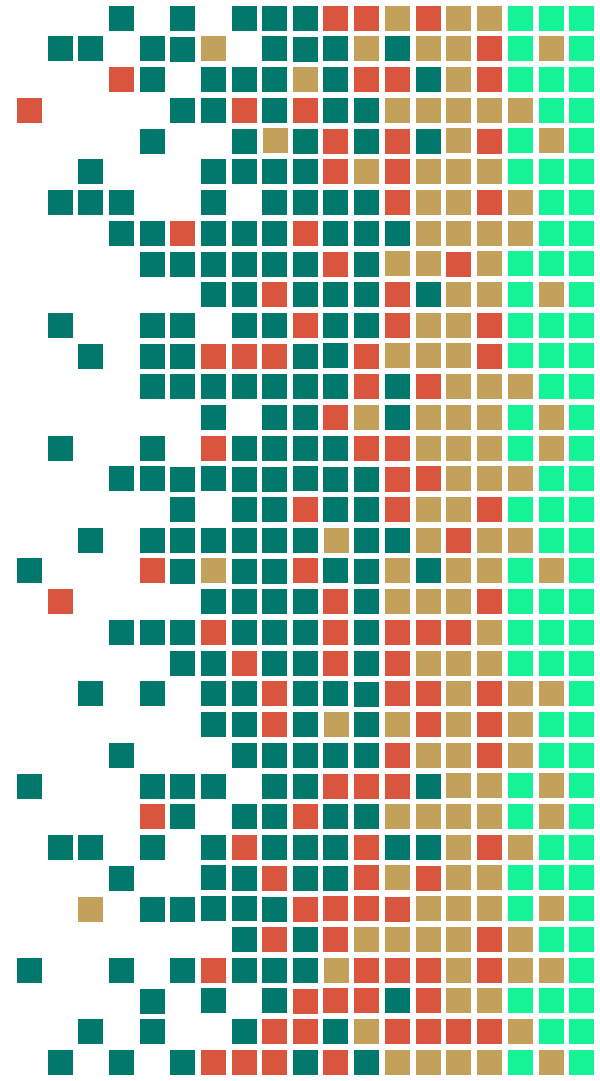




# Expectations before Starting

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

### 3. Dclivcrublc Thrcc



# The Attacks

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

# Pusssword 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.

```
(kali@kali)~/acim-WebDenial-main
$ hydra -l root -P /home/kali/rockyoy.txt mysql://localhost
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or
secret service organizations, or for illegal purposes (this is non-binding, these *** ig
nore laws and ethics anyway).

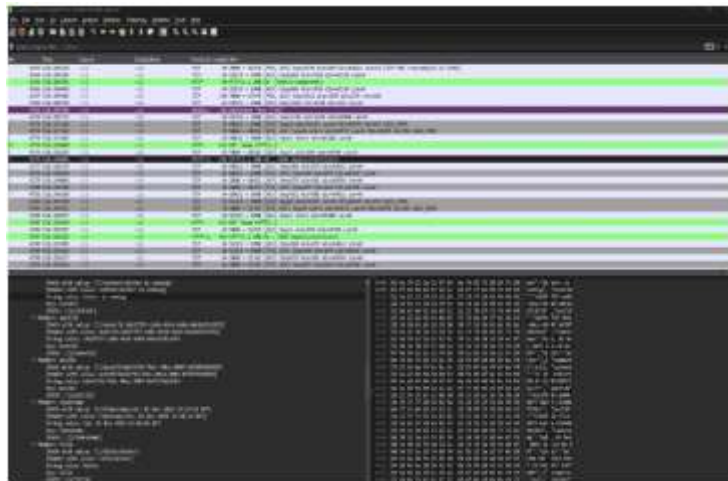
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-10-30 22:04:24
[INFO] Reduced number of tasks to 4 (mysql does not like many parallel connections)
[DATA] max 4 tasks per 1 server, overall 4 tasks, 30001 login tries (l:1/p:30001), ~7501
tries per task
[DATA] attacking mysql://localhost:3306/
[STATUS] 9901.00 tries/min, 9901 tries in 00:01h, 20100 to do in 00:03h, 4 active
[STATUS] 9805.00 tries/min, 19610 tries in 00:02h, 10391 to do in 00:02h, 4 active
[STATUS] 9725.67 tries/min, 29177 tries in 00:03h, 824 to do in 00:01h, 4 active
[3306][mysql] host: localhost login: root password: ilovesecurity
1 of 1 target successfully completed, 1 valid password found
[WARNING] Writing restore file because 1 final worker threads did not complete until end.
[ERROR] 1 target did not resolve or could not be connected
[ERROR] 0 target did not complete
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-10-30 22:07:30

(kali@kali)~/acim-WebDenial-main
$
```

# Puckct Sniffing

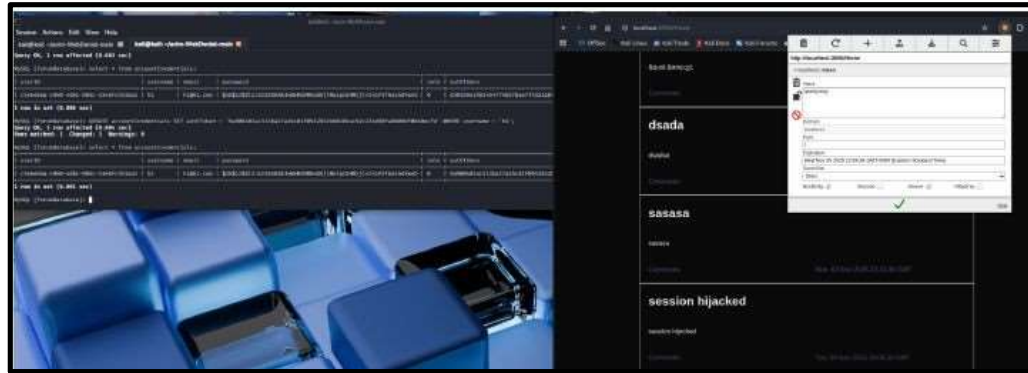
We also performed packet sniffing on the application using Wireshark. While the standard handshakes were encrypted through the QUIC protocol, we found that when a user created a post, the transmitted data could still be captured in plaintext.

46.825173236	10.0.2.3	10.0.2.15	DNS	332 Standard query response ExecC A passwords!eakcheck-pa.googleapis.com A 142.250.89.74 A 142.250
50.832148559	10.0.2.3	10.0.2.15	DNS	298 Standard query response ExecC AAAA passwords!eakcheck-pa.googleapis.com AAAA 2007:f8b9:4b96:82
60.832671526	10.0.2.3	10.0.2.15	DNS	156 Standard query response ExecC HTTPS passwords!eakcheck-pa.googleapis.com XA nci.google.com
70.833152582	10.0.2.15	142.250.89.74	QUIC	1292 Initial, DCID=62898ef48b549648, PKN: 1, CRYPTO, PING, CRYPTO, PING, CRYPTO, PING, CRYPTO, CRYPTO, CRYPTO
80.833294638	10.0.2.15	142.250.89.74	QUIC	1292 Initial, DCID=62898ef48b549648, PKN: 2, PADDING, PING, PADDING, CRYPTO, PADDING, CRYPTO, CRYPTO
90.854671327	142.250.89.74	10.0.2.15	QUIC	82 Initial, SCID=e2898ef48b549648, PKN: 1, ACK
100.854671291	142.250.89.74	10.0.2.15	QUIC	1292 Initial, SCID=e2898ef48b549648, PKN: 2, ACK, PADDING
110.854671320	142.250.89.74	10.0.2.15	QUIC	1292 Initial, SCID=e2898ef48b549648, PKN: 3, CRYPTO, PADDING
120.855074849	142.250.89.74	10.0.2.15	QUIC	1292 Initial, SCID=e2898ef48b549648, PKN: 4, CRYPTO, PADDING
130.855294810	10.0.2.15	142.250.89.74	QUIC	1292 Initial, DCID=e2898ef48b549648, PKN: 3, ACK, PADDING
140.860703915	142.250.89.74	10.0.2.15	QUIC	1292 Handshake, SCID=e2898ef48b549648
150.860704000	142.250.89.74	10.0.2.15	QUIC	1292 Handshake, SCID=e2898ef48b549648
160.860704006	142.250.89.74	10.0.2.15	QUIC	1292 Handshake, SCID=e2898ef48b549648
170.860806684	10.0.2.15	142.250.89.74	QUIC	81 Handshake, DCID=e2898ef48b549648
180.863899578	10.0.2.15	142.250.89.74	QUIC	82 Handshake, DCID=e2898ef48b549648
190.875333903	142.250.89.74	10.0.2.15	QUIC	116 Protected Payload (KPE), DCID=e2898ef48b549648
200.876602248	10.0.2.15	142.250.89.74	QUIC	131 Handshake, DCID=e2898ef48b549648
210.876505729	10.0.2.15	142.250.89.74	QUIC	116 Protected Payload (KPE), DCID=e2898ef48b549648
220.877117884	10.0.2.15	142.250.89.74	QUIC	453 Protected Payload (KPE), DCID=e2898ef48b549648
230.887633744	142.250.89.74	10.0.2.15	QUIC	1023 Protected Payload (KPE)
240.887633863	142.250.89.74	10.0.2.15	QUIC	153 Protected Payload (KPE)



# Session Hijacking

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.



# Denial of Service

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.

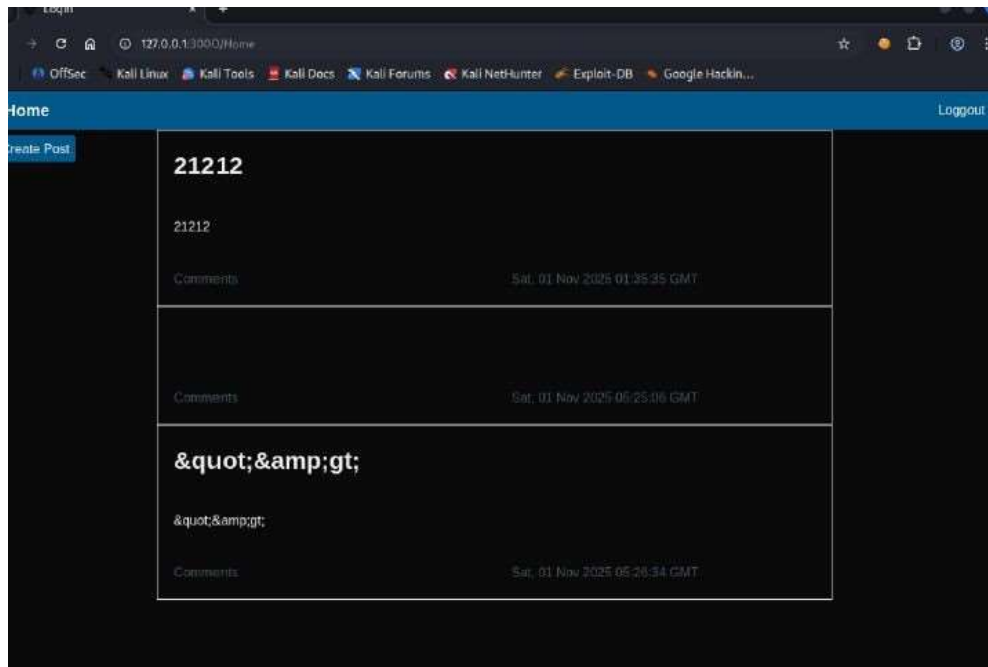


```
➤ POST http://localhost:5000/login 401 (UNAUTHORIZED) VM416:6
10.0.0.7.17 401 VM416:16
➤ POST http://localhost:5000/login 401 (UNAUTHORIZED) VM418:6
10.0.0.7.18 401 VM416:16
➤ POST http://localhost:5000/login 401 (UNAUTHORIZED) VM416:6
10.0.0.7.19 401 VM416:16
➤ POST http://localhost:5000/login 401 (UNAUTHORIZED) VM410:2
10.0.0.7.20 401 VM416:16
➤ POST http://localhost:5000/login 401 (UNAUTHORIZED) VM416:6
10.0.0.7.21 401 VM416:16
➤ POST http://localhost:5000/login 401 (UNAUTHORIZED) VM416:6
10.0.0.7.22 401 VM416:16
➤ POST http://localhost:5000/login 401 (UNAUTHORIZED) VM418:6
10.0.0.7.23 401 VM416:16
➤ POST http://localhost:5000/login 401 (UNAUTHORIZED) VM416:6
10.0.0.7.24 401 VM416:16
```

→ NO more error 429

# Cross Site 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 Chunnc1

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.

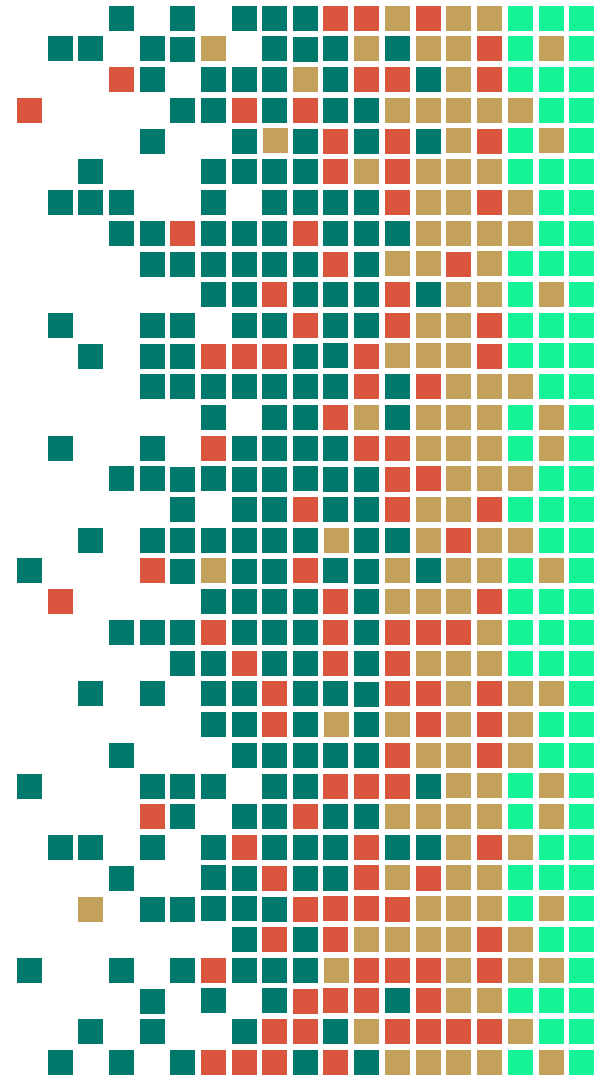
```
partho@LAPTOP-MMT3N4B ~/acim-WebDenial (main)> for i in (seq 1 10); curl -sS -m 5 -o /dev/null -w "%{time_total}\n" -H "Content-Type: application/json" -d '{"username": "Dummy", "password": "Dummy77?"}' http://localhost:5000/Login; end
0.277939
0.285525
0.274266
0.286068
0.260663
0.262412
0.255141
0.280442
0.277949
0.274513
```

```
partho@LAPTOP-MMT3N4B ~/acim-WebDenial (main)> for i in (seq 1 10); curl -sS -m 5 -o /dev/null -w "%{time_total}\n" -H "Content-Type: application/json" -d '{"username": "no_such_user_abcdef", "password": "wrong"}' http://localhost:5000/Login; end
0.048369
0.037941
0.039583
0.039624
0.036401
0.039365
0.039027
0.038112
0.039141
0.039270
```

# Experiences gained and Lessons Learned

- 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 didn't

## 4. Dclivcrublc Four



# Fixing vulncrubitics

- 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

## Register

[Home](#) | [Login](#)

Username: hi2

Email: hi2@gmail.com

Password: ...



Please match the requested format.

Must be at least 8 characters, include 1 uppercase letter, 1 number, and 1 special character.

[illegible]

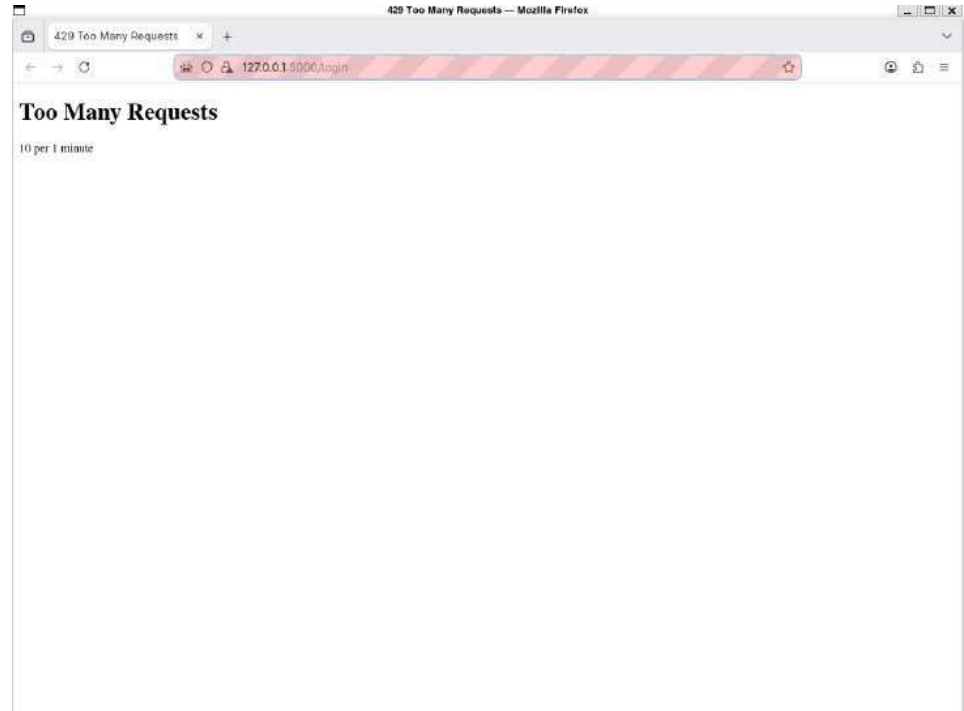
# Puckct Snivfing

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



# Denial of Service (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.



# Session Hijacking

- 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.



The screenshot shows a browser's developer console with the following content:

```
top Filter  
> doc  
Uncaught ReferenceError: doc is not defined  
    at <anonymous>:1:1  
> clear  
< f clear() { [native code] }  
> cls  
Uncaught ReferenceError: cls is not defined  
    at <anonymous>:1:1  
> document.cookie  
< 'ajs_anonymous_id=4f25d448-e689-4d71-bd6f-85fa0f1a63ef'  
> |
```



# Cross-Site 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.



# Overall Learning

- 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!**

