



AUDIT REPORT

October 2025

For



MNEE

Table of Content

Executive Summary	04
Number of Security Issues per Severity	05
Summary of Issues	06
Checked Vulnerabilities	08
Techniques and Methods	09
Types of Severity	10
Types of Issues	11
Severity Matrix	12
■ High Severity Issues	13
1. Authentication Issues	13
2. Broken Access Control	16
3. Broken Access Control	18
4. Amount Value is Not Validated During Burn	19
5. Unvalidated Blacklisted Request Cancellation	20
6. Broken Access Control	21
7. Unauthorised Refund	22
■ Medium Severity Issues	24
8. Usage of weak hashing library (MD5)	24
9. Insecure Input Validation	25
10. No Rate Limit	26

 Low Severity Issues	27
11. Leakage of sensitive information in logger message	27
12. Middleware Does not Handle Token Expiration	29
13. User Enumeration	30
14. Weak Password Check	31
15. Improper Error Handling	32
16. Outdated Server Components	33
Closing Summary & Disclaimer	34

Executive Summary

Project Name	MNEE
Project URL	https://github.com/rockwalletcode/mnee-cosigner https://github.com/rockwalletcode/mnee-portal
Overview	MNEE (pronounced “money”) is a fast, USD-backed stablecoin designed for digital payments, gaming, remittances, and more. Fully collateralized 1:1 with U.S. Treasury bills and cash equivalents, MNEE ensures stability and transparency through regular audits. It operates on the high-speed 1Sat Ordinals protocol , offering near-zero fees and instant finality. MNEE is regulated in Antigua with full AML/KYC compliance.
Review 1	11th February 2025 - 25th March 2025
Updated Code Received	7th October 2025
Review 2	28th October 2025 - 29th October 2025

Verify the Authenticity of Report on QuillAudits Leaderboard:

<https://www.quillaudits.com/leaderboard>

Number of Issues per Severity



Critical	0 (0.0%)
High	7 (43.8%)
Medium	3 (18.7%)
Low	6 (37.5%)
Informational	0 (0.0%)

Issues	Severity				
	Critical	High	Medium	Low	Informational
Open	0	0	0	0	0
Acknowledged	0	1	1	0	0
Partially Resolved	0	0	0	0	0
Resolved	0	6	2	6	0

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Authentication Issues	High	Resolved
2	Broken Access Control	High	Resolved
3	Broken Access Control	High	Acknowledged
4	Amount Value is Not Validated During Burn	High	Resolved
5	Unvalidated Blacklisted Request Cancellation	High	Resolved
6	Broken Access Control	High	Resolved
7	Unauthorised Refund	High	Resolved
8	Usage of weak hashing library (MD5)	Medium	Resolved
9	Insecure Input Validation	Medium	Resolved
10	No Rate Limit	Medium	Acknowledged
11	Leakage of sensitive information in logger message	Low	Resolved

Issue No.	Issue Title	Severity	Status
12	Middleware Does not Handle Token Expiration	Low	Resolved
13	User Enumeration	Low	Resolved
14	Weak Password Check	Low	Resolved
15	Improper Error Handling	Low	Resolved
16	Outdated Server Components	Low	Resolved

Checked Vulnerabilities

Improper Authentication

Improper Resource Usage

Improper Authorization

Insecure File Uploads

Insecure Direct Object References

Client-Side Validation Issues

Rate Limit

Input Validation

Injection Attacks

Cross-Site Scripting (XSS)

Cross-Site Request Forgery

Security Misconfiguration

Broken Access Controls

Insecure Cryptographic Storage

Insufficient Cryptography

Insufficient Session Expiration

Insufficient Transport Layer Protection

Unvalidated Redirects and Forwards

Information Leakage

Broken Authentication and Session Management

Denial of Service (DoS) Attacks

Malware

Third-Party Components

And More..

Techniques and Methods

Throughout the pentest of application, care was taken to ensure:

- Information gathering – Using OSINT tools information concerning the web architecture, information leakage, web service integration, and gathering other associated information related to web server & web services.
- Using Automated tools approach for Pentest like Nessus, Acunetix etc.
- Platform testing and configuration
- Error handling and data validation testing
- Encryption-related protection testing
- Client-side and business logic testing

Tools and Platforms used for Pentest:

Burp Suite

Acunetix

Nmap

DNSenum

Neucli

Metasploit

Dirbuster

Nabbu

Horusec

SQLMap

Turbo Intruder

Postman

Netcat

Nessus

And Many more..

Types of Severity

Every issue in this report has been assigned to a severity level. There are five levels of severity, and each of them has been explained below.

Critical: Immediate and Catastrophic Impact

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

High (H): Significant Risk of Major Loss or Compromise

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

Medium (M): Potential for Moderate Harm Under Specific Circumstances

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

Low (L): Minor Imperfections with Limited Repercussions

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

Informational (I): Opportunities for Improvement, Not Immediate Risks

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.

Types of Issues

Open	Security vulnerabilities identified that must be resolved and are currently unresolved.	Resolved	These are the issues identified in the initial audit and have been successfully fixed.
Acknowledged	Vulnerabilities which have been acknowledged but are yet to be resolved.	Partially Resolved	Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Severity Matrix

		Impact		
		High	Medium	Low
Likelihood	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

High Severity Issues

Authentication Issues

Resolved

Description

The application has multiple authentication based issues.

1. Lack of Rate Limiting & Brute Force Protection: The code does not implement rate limiting or account lockout mechanisms, which makes it susceptible to brute-force attacks.
2. No MFA (Multi-Factor Authentication) Support: The authentication process only relies on email and password, which is not sufficient against credential stuffing attacks.
3. Authentication and Authorization Cookie: The application has not implemented any Authorization cookies or token. This can allow unauthorized users to interact with APIs by directly sending the API a request to carry out an action. Since there is no authorization token, all requests are expected to be accepted by API without verification.
5. CSRF Protection: The application has not implemented CSRF token. An anti-CSRF token should be implemented to block users from attempting (1) brute-force attacks and also perform a CSRF attack by crafting special requests.

Vulnerable File

mnee-portal-master/src/utils/auth.ts

mnee-portal-master/src/utils/api.ts

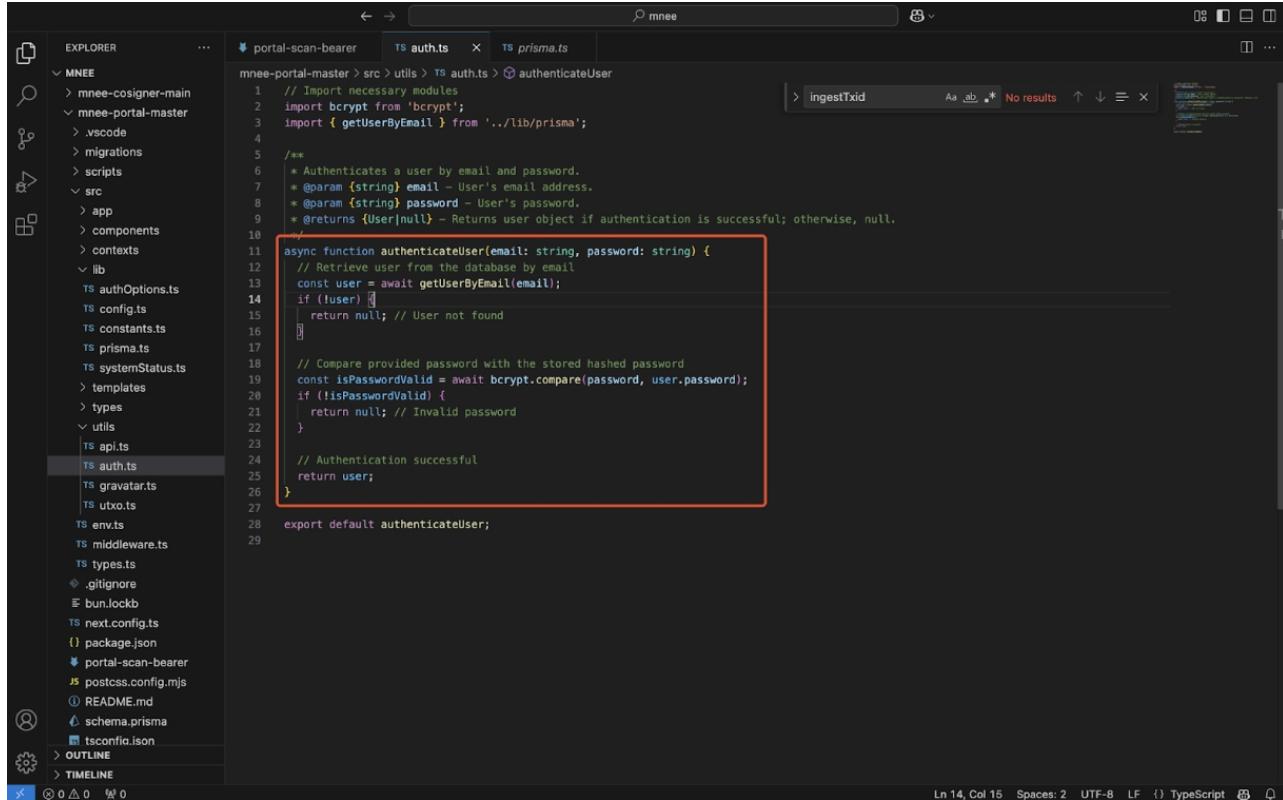
Impact

1. Attackers can brute-force username and password values

Recommendation

1. Implement rate limiting to prevent brute-force attacks on authentication mechanisms
2. Implement rate limiting on all API endpoints. You can also implement an anti-CSRF token to prevent brute-force attacks as a add-on security mechanism
3. Implement authentication/authorization cookies to prevent Broken Access Control attacks
4. Implement a check against IDOR vulnerabilities (application wide). The function should check if the corresponding value passed in the request body belongs to the person who is currently authenticated and if he is authorized to execute the request with the data parsed.

POC

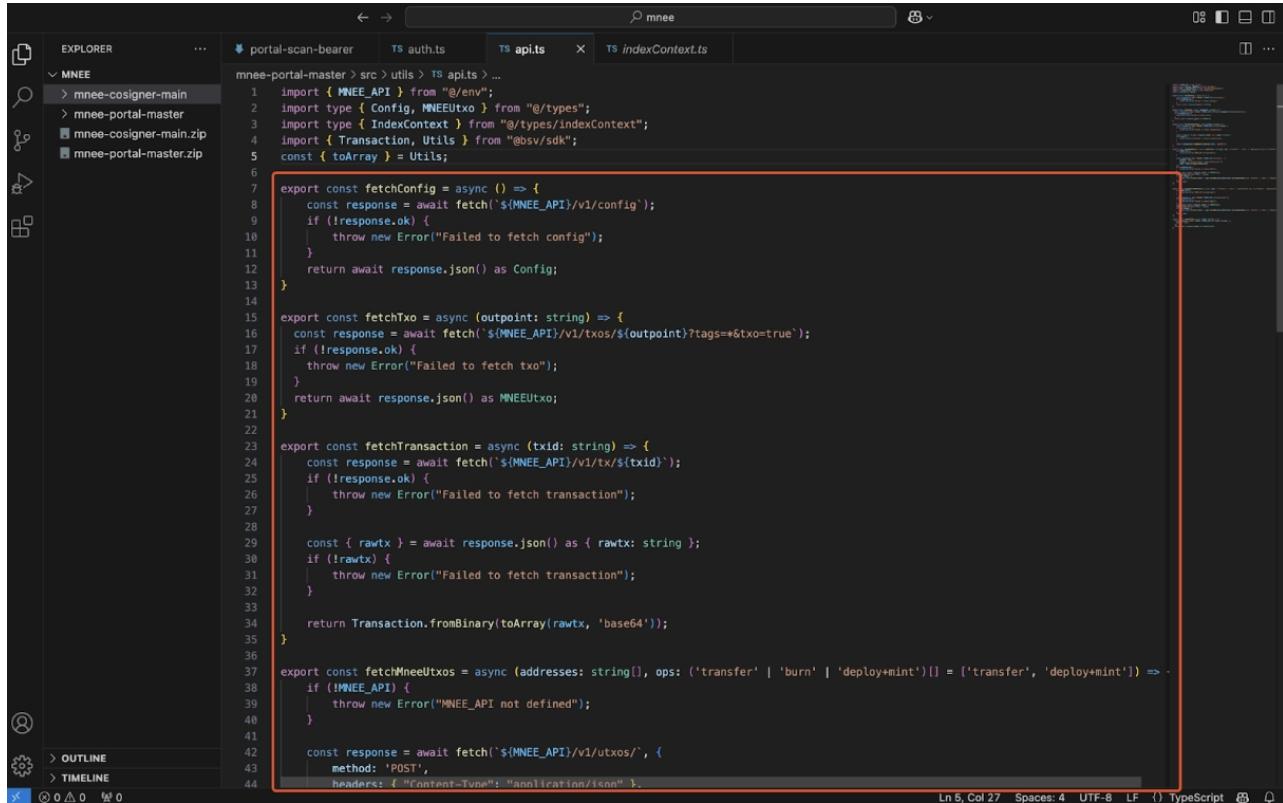


```

EXPLORER      TS auth.ts      TS prismats.ts
mnee-portal-master > src > utils > TS auth.ts > authenticateUser
1 // Import necessary modules
2 import bcrypt from 'bcrypt';
3 import { getUserByEmail } from '../lib/prisma';
4
5 /**
6 * Authenticates a user by email and password.
7 * @param {string} email - User's email address.
8 * @param {string} password - User's password.
9 * @returns {User|null} - Returns user object if authentication is successful; otherwise, null.
10 */
11 async function authenticateUser(email: string, password: string) {
12   // Retrieve user from the database by email
13   const user = await getUserByEmail(email);
14   if (!user) {
15     return null; // User not found
16   }
17
18   // Compare provided password with the stored hashed password
19   const isPasswordValid = await bcrypt.compare(password, user.password);
20   if (!isPasswordValid) {
21     return null; // Invalid password
22   }
23
24   // Authentication successful
25   return user;
26 }
27
28 export default authenticateUser;
29

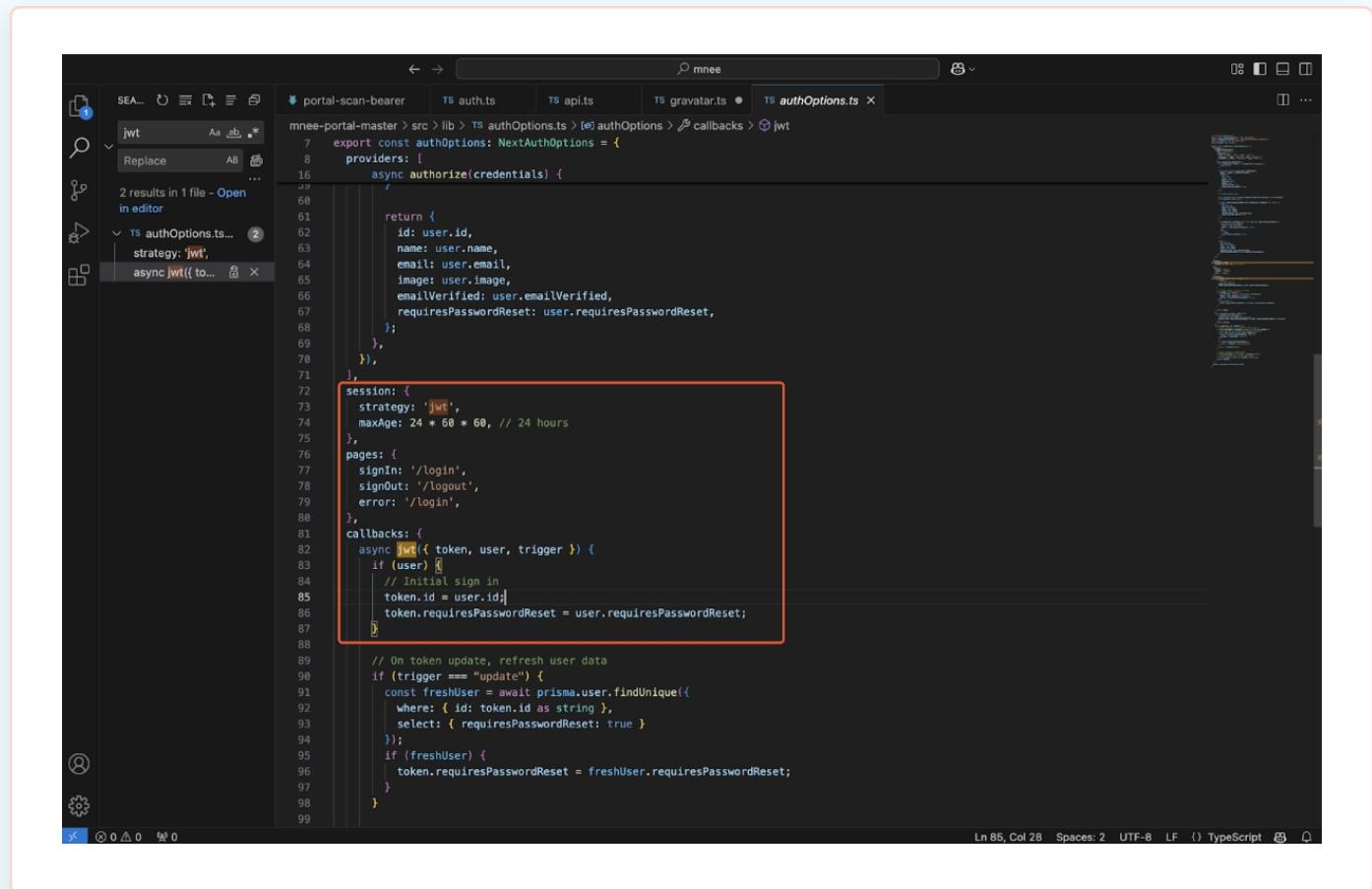
```

Ln 14, Col 15 Spaces: 2 UTF-8 LF () TypeScript



```

EXPLORER      TS auth.ts      TS api.ts      TS indexContext.ts
mnee-portal-master > src > utils > TS api.ts > ...
1 import { MNEE_API } from '@env';
2 import type { Config, MNEEUtxo } from "@types";
3 import type { IndexContext } from "@types/indexContext";
4 import { Transaction, Utils } from "@osv/sdk";
5 const { toArray } = Utils;
6
7 export const fetchConfig = async () => {
8   const response = await fetch(`${MNEE_API}/v1/config`);
9   if (!response.ok) {
10     throw new Error("Failed to fetch config");
11   }
12   return await response.json() as Config;
13 }
14
15 export const fetchTxo = async (outpoint: string) => {
16   const response = await fetch(`.${MNEE_API}/v1/txos/${outpoint}?tags=&txo=true`);
17   if (!response.ok) {
18     throw new Error("Failed to fetch txo");
19   }
20   return await response.json() as MNEEUtxo;
21 }
22
23 export const fetchTransaction = async (txid: string) => {
24   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}`);
25   if (!response.ok) {
26     throw new Error("Failed to fetch transaction");
27   }
28
29   const { rawtx } = await response.json() as { rawtx: string };
30   if (!rawtx) {
31     throw new Error("Failed to fetch transaction");
32   }
33
34   return Transaction.fromBinary(toArray(rawtx, 'base64'));
35 }
36
37 export const fetchMneeUtxos = async (addresses: string[], ops: ['transfer' | 'burn' | 'deploy+mint'][] = ['transfer', 'deploy+mint']) => {
38   if (!MNEE_API) {
39     throw new Error("MNEE_API not defined");
40   }
41
42   const response = await fetch(`.${MNEE_API}/v1/utxos/`, {
43     method: 'POST',
44     headers: { 'Content-Type': "application/json" },
45     body: JSON.stringify({ addresses, ops })
46   });
47
48   if (!response.ok) {
49     throw new Error(`Failed to fetch utxos: ${response.statusText}`);
50   }
51
52   return response.json() as MNEEUtxo[];
53 }
54
55 export const signTx = async (tx: Transaction, privateKey: string) => {
56   const signedTx = tx.sign(privateKey);
57   const serializedTx = signedTx.serialize();
58   const hexTx = Buffer.from(serializedTx).toString('hex');
59
60   const response = await fetch(`.${MNEE_API}/v1/tx`, {
61     method: 'POST',
62     headers: { 'Content-Type': "application/json" },
63     body: JSON.stringify({ tx: hexTx })
64   });
65
66   if (!response.ok) {
67     throw new Error(`Failed to sign tx: ${response.statusText}`);
68   }
69
70   return response.json() as MNEEUtxo;
71 }
72
73 export const broadcastTx = async (tx: Transaction) => {
74   const serializedTx = tx.serialize();
75   const hexTx = Buffer.from(serializedTx).toString('hex');
76
77   const response = await fetch(`.${MNEE_API}/v1/tx/broadcast`, {
78     method: 'POST',
79     headers: { 'Content-Type': "application/json" },
80     body: JSON.stringify({ tx: hexTx })
81   });
82
83   if (!response.ok) {
84     throw new Error(`Failed to broadcast tx: ${response.statusText}`);
85   }
86
87   return response.json() as MNEEUtxo;
88 }
89
90 export const getTxStatus = async (txid: string) => {
91   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}`);
92   if (!response.ok) {
93     throw new Error("Failed to get tx status");
94   }
95   return response.json() as MNEEUtxo;
96 }
97
98 export const getUtxos = async (addresses: string[]) => {
99   const response = await fetch(`.${MNEE_API}/v1/utxos`, {
100    method: 'GET',
101    headers: { 'Content-Type': "application/json" },
102    body: JSON.stringify({ addresses })
103  });
104
105  if (!response.ok) {
106    throw new Error(`Failed to get utxos: ${response.statusText}`);
107  }
108
109  return response.json() as MNEEUtxo[];
110 }
111
112 export const getTx = async (txid: string) => {
113   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}`);
114   if (!response.ok) {
115     throw new Error("Failed to get tx");
116   }
117   return response.json() as Transaction;
118 }
119
120 export const getBlock = async (blockHash: string) => {
121   const response = await fetch(`.${MNEE_API}/v1/block/${blockHash}`);
122   if (!response.ok) {
123     throw new Error("Failed to get block");
124   }
125   return response.json() as Block;
126 }
127
128 export const getBlocks = async (startBlock: string, endBlock: string) => {
129   const response = await fetch(`.${MNEE_API}/v1/blocks?start=${startBlock}&end=${endBlock}`);
130   if (!response.ok) {
131     throw new Error("Failed to get blocks");
132   }
133   return response.json() as Block[];
134 }
135
136 export const getLogs = async (txid: string) => {
137   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/logs`);
138   if (!response.ok) {
139     throw new Error("Failed to get logs");
140   }
141   return response.json() as Log[];
142 }
143
144 export const getRawTx = async (txid: string) => {
145   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw`);
146   if (!response.ok) {
147     throw new Error("Failed to get raw tx");
148   }
149   return response.json() as string;
150 }
151
152 export const getRawLogs = async (txid: string) => {
153   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs`);
154   if (!response.ok) {
155     throw new Error("Failed to get raw logs");
156   }
157   return response.json() as string[];
158 }
159
160 export const getRawBlock = async (blockHash: string) => {
161   const response = await fetch(`.${MNEE_API}/v1/block/${blockHash}/raw`);
162   if (!response.ok) {
163     throw new Error("Failed to get raw block");
164   }
165   return response.json() as string;
166 }
167
168 export const getRawBlocks = async (startBlock: string, endBlock: string) => {
169   const response = await fetch(`.${MNEE_API}/v1/blocks?start=${startBlock}&end=${endBlock}/raw`);
170   if (!response.ok) {
171     throw new Error("Failed to get raw blocks");
172   }
173   return response.json() as string[];
174 }
175
176 export const getRawLog = async (txid: string, logIndex: number) => {
177   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}`);
178   if (!response.ok) {
179     throw new Error("Failed to get raw log");
180   }
181   return response.json() as string;
182 }
183
184 export const getRawLogsForTx = async (txid: string) => {
185   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs`);
186   if (!response.ok) {
187     throw new Error("Failed to get raw logs for tx");
188   }
189   return response.json() as string[];
190 }
191
192 export const getRawLogsForBlock = async (blockHash: string) => {
193   const response = await fetch(`.${MNEE_API}/v1/block/${blockHash}/raw/logs`);
194   if (!response.ok) {
195     throw new Error("Failed to get raw logs for block");
196   }
197   return response.json() as string[];
198 }
199
200 export const getRawLogsForRange = async (startBlock: string, endBlock: string) => {
201   const response = await fetch(`.${MNEE_API}/v1/blocks?start=${startBlock}&end=${endBlock}/raw/logs`);
202   if (!response.ok) {
203     throw new Error("Failed to get raw logs for range");
204   }
205   return response.json() as string[];
206 }
207
208 export const getRawLogForIndex = async (txid: string, logIndex: number) => {
209   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}`);
210   if (!response.ok) {
211     throw new Error("Failed to get raw log for index");
212   }
213   return response.json() as string;
214 }
215
216 export const getRawLogsForTxIndex = async (txid: string, logIndex: number) => {
217   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}`);
218   if (!response.ok) {
219     throw new Error("Failed to get raw logs for tx index");
220   }
221   return response.json() as string[];
222 }
223
224 export const getRawLogsForBlockIndex = async (blockHash: string, logIndex: number) => {
225   const response = await fetch(`.${MNEE_API}/v1/block/${blockHash}/raw/logs/${logIndex}`);
226   if (!response.ok) {
227     throw new Error("Failed to get raw logs for block index");
228   }
229   return response.json() as string[];
230 }
231
232 export const getRawLogsForRangeIndex = async (startBlock: string, endBlock: string, logIndex: number) => {
233   const response = await fetch(`.${MNEE_API}/v1/blocks?start=${startBlock}&end=${endBlock}/raw/logs/${logIndex}`);
234   if (!response.ok) {
235     throw new Error("Failed to get raw logs for range index");
236   }
237   return response.json() as string[];
238 }
239
240 export const getRawLogForIndexIndex = async (txid: string, logIndex: number, logIndex2: number) => {
241   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}`);
242   if (!response.ok) {
243     throw new Error("Failed to get raw log for index index");
244   }
245   return response.json() as string;
246 }
247
248 export const getRawLogsForTxIndexIndex = async (txid: string, logIndex: number, logIndex2: number) => {
249   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}`);
250   if (!response.ok) {
251     throw new Error("Failed to get raw logs for tx index index");
252   }
253   return response.json() as string[];
254 }
255
256 export const getRawLogsForBlockIndexIndex = async (blockHash: string, logIndex: number, logIndex2: number) => {
257   const response = await fetch(`.${MNEE_API}/v1/block/${blockHash}/raw/logs/${logIndex}/${logIndex2}`);
258   if (!response.ok) {
259     throw new Error("Failed to get raw logs for block index index");
260   }
261   return response.json() as string[];
262 }
263
264 export const getRawLogsForRangeIndexIndex = async (startBlock: string, endBlock: string, logIndex: number, logIndex2: number) => {
265   const response = await fetch(`.${MNEE_API}/v1/blocks?start=${startBlock}&end=${endBlock}/raw/logs/${logIndex}/${logIndex2}`);
266   if (!response.ok) {
267     throw new Error("Failed to get raw logs for range index index");
268   }
269   return response.json() as string[];
270 }
271
272 export const getRawLogForIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number) => {
273   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}`);
274   if (!response.ok) {
275     throw new Error("Failed to get raw log for index index index");
276   }
277   return response.json() as string;
278 }
279
280 export const getRawLogsForTxIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number) => {
281   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}`);
282   if (!response.ok) {
283     throw new Error("Failed to get raw logs for tx index index index");
284   }
285   return response.json() as string[];
286 }
287
288 export const getRawLogsForBlockIndexIndexIndex = async (blockHash: string, logIndex: number, logIndex2: number, logIndex3: number) => {
289   const response = await fetch(`.${MNEE_API}/v1/block/${blockHash}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}`);
290   if (!response.ok) {
291     throw new Error("Failed to get raw logs for block index index index");
292   }
293   return response.json() as string[];
294 }
295
296 export const getRawLogsForRangeIndexIndexIndex = async (startBlock: string, endBlock: string, logIndex: number, logIndex2: number, logIndex3: number) => {
297   const response = await fetch(`.${MNEE_API}/v1/blocks?start=${startBlock}&end=${endBlock}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}`);
298   if (!response.ok) {
299     throw new Error("Failed to get raw logs for range index index index");
300   }
301   return response.json() as string[];
302 }
303
304 export const getRawLogForIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number) => {
305   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}`);
306   if (!response.ok) {
307     throw new Error("Failed to get raw log for index index index index");
308   }
309   return response.json() as string;
310 }
311
312 export const getRawLogsForTxIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number) => {
313   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}`);
314   if (!response.ok) {
315     throw new Error("Failed to get raw logs for tx index index index index");
316   }
317   return response.json() as string[];
318 }
319
320 export const getRawLogsForBlockIndexIndexIndexIndex = async (blockHash: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number) => {
321   const response = await fetch(`.${MNEE_API}/v1/block/${blockHash}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}`);
322   if (!response.ok) {
323     throw new Error("Failed to get raw logs for block index index index index");
324   }
325   return response.json() as string[];
326 }
327
328 export const getRawLogsForRangeIndexIndexIndexIndex = async (startBlock: string, endBlock: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number) => {
329   const response = await fetch(`.${MNEE_API}/v1/blocks?start=${startBlock}&end=${endBlock}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}`);
330   if (!response.ok) {
331     throw new Error("Failed to get raw logs for range index index index index");
332   }
333   return response.json() as string[];
334 }
335
336 export const getRawLogForIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number) => {
337   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}`);
338   if (!response.ok) {
339     throw new Error("Failed to get raw log for index index index index index");
340   }
341   return response.json() as string;
342 }
343
344 export const getRawLogsForTxIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number) => {
345   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}`);
346   if (!response.ok) {
347     throw new Error("Failed to get raw logs for tx index index index index index");
348   }
349   return response.json() as string[];
350 }
351
352 export const getRawLogsForBlockIndexIndexIndexIndexIndex = async (blockHash: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number) => {
353   const response = await fetch(`.${MNEE_API}/v1/block/${blockHash}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}`);
354   if (!response.ok) {
355     throw new Error("Failed to get raw logs for block index index index index index");
356   }
357   return response.json() as string[];
358 }
359
360 export const getRawLogsForRangeIndexIndexIndexIndexIndex = async (startBlock: string, endBlock: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number) => {
361   const response = await fetch(`.${MNEE_API}/v1/blocks?start=${startBlock}&end=${endBlock}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}`);
362   if (!response.ok) {
363     throw new Error("Failed to get raw logs for range index index index index index");
364   }
365   return response.json() as string[];
366 }
367
368 export const getRawLogForIndexIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number) => {
369   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}`);
370   if (!response.ok) {
371     throw new Error("Failed to get raw log for index index index index index index");
372   }
373   return response.json() as string;
374 }
375
376 export const getRawLogsForTxIndexIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number) => {
377   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}`);
378   if (!response.ok) {
379     throw new Error("Failed to get raw logs for tx index index index index index index");
380   }
381   return response.json() as string[];
382 }
383
384 export const getRawLogsForBlockIndexIndexIndexIndexIndexIndex = async (blockHash: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number) => {
385   const response = await fetch(`.${MNEE_API}/v1/block/${blockHash}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}`);
386   if (!response.ok) {
387     throw new Error("Failed to get raw logs for block index index index index index index");
388   }
389   return response.json() as string[];
390 }
391
392 export const getRawLogsForRangeIndexIndexIndexIndexIndexIndex = async (startBlock: string, endBlock: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number) => {
393   const response = await fetch(`.${MNEE_API}/v1/blocks?start=${startBlock}&end=${endBlock}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}`);
394   if (!response.ok) {
395     throw new Error("Failed to get raw logs for range index index index index index index");
396   }
397   return response.json() as string[];
398 }
399
400 export const getRawLogForIndexIndexIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number) => {
401   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}`);
402   if (!response.ok) {
403     throw new Error("Failed to get raw log for index index index index index index index");
404   }
405   return response.json() as string;
406 }
407
408 export const getRawLogsForTxIndexIndexIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number) => {
409   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}`);
410   if (!response.ok) {
411     throw new Error("Failed to get raw logs for tx index index index index index index index");
412   }
413   return response.json() as string[];
414 }
415
416 export const getRawLogsForBlockIndexIndexIndexIndexIndexIndexIndex = async (blockHash: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number) => {
417   const response = await fetch(`.${MNEE_API}/v1/block/${blockHash}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}`);
418   if (!response.ok) {
419     throw new Error("Failed to get raw logs for block index index index index index index index");
420   }
421   return response.json() as string[];
422 }
423
424 export const getRawLogsForRangeIndexIndexIndexIndexIndexIndexIndex = async (startBlock: string, endBlock: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number) => {
425   const response = await fetch(`.${MNEE_API}/v1/blocks?start=${startBlock}&end=${endBlock}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}`);
426   if (!response.ok) {
427     throw new Error("Failed to get raw logs for range index index index index index index index");
428   }
429   return response.json() as string[];
430 }
431
432 export const getRawLogForIndexIndexIndexIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number) => {
433   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}`);
434   if (!response.ok) {
435     throw new Error("Failed to get raw log for index index index index index index index index");
436   }
437   return response.json() as string;
438 }
439
440 export const getRawLogsForTxIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number) => {
441   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}`);
442   if (!response.ok) {
443     throw new Error("Failed to get raw logs for tx index index index index index index index index");
444   }
445   return response.json() as string[];
446 }
447
448 export const getRawLogsForBlockIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (blockHash: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number) => {
449   const response = await fetch(`.${MNEE_API}/v1/block/${blockHash}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}`);
450   if (!response.ok) {
451     throw new Error("Failed to get raw logs for block index index index index index index index index");
452   }
453   return response.json() as string[];
454 }
455
456 export const getRawLogsForRangeIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (startBlock: string, endBlock: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number) => {
457   const response = await fetch(`.${MNEE_API}/v1/blocks?start=${startBlock}&end=${endBlock}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}`);
458   if (!response.ok) {
459     throw new Error("Failed to get raw logs for range index index index index index index index index");
460   }
461   return response.json() as string[];
462 }
463
464 export const getRawLogForIndexIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number, logIndex9: number) => {
465   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}/${logIndex9}`);
466   if (!response.ok) {
467     throw new Error("Failed to get raw log for index index index index index index index index index");
468   }
469   return response.json() as string;
470 }
471
472 export const getRawLogsForTxIndexIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number, logIndex9: number) => {
473   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}/${logIndex9}`);
474   if (!response.ok) {
475     throw new Error("Failed to get raw logs for tx index index index index index index index index index");
476   }
477   return response.json() as string[];
478 }
479
480 export const getRawLogsForBlockIndexIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (blockHash: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number, logIndex9: number) => {
481   const response = await fetch(`.${MNEE_API}/v1/block/${blockHash}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}/${logIndex9}`);
482   if (!response.ok) {
483     throw new Error("Failed to get raw logs for block index index index index index index index index index");
484   }
485   return response.json() as string[];
486 }
487
488 export const getRawLogsForRangeIndexIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (startBlock: string, endBlock: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number, logIndex9: number) => {
489   const response = await fetch(`.${MNEE_API}/v1/blocks?start=${startBlock}&end=${endBlock}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}/${logIndex9}`);
490   if (!response.ok) {
491     throw new Error("Failed to get raw logs for range index index index index index index index index index");
492   }
493   return response.json() as string[];
494 }
495
496 export const getRawLogForIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number, logIndex9: number, logIndex10: number) => {
497   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}/${logIndex9}/${logIndex10}`);
498   if (!response.ok) {
499     throw new Error("Failed to get raw log for index index index index index index index index index index");
500   }
501   return response.json() as string;
502 }
503
504 export const getRawLogsForTxIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number, logIndex9: number, logIndex10: number) => {
505   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}/${logIndex9}/${logIndex10}`);
506   if (!response.ok) {
507     throw new Error("Failed to get raw logs for tx index index index index index index index index index index");
508   }
509   return response.json() as string[];
510 }
511
512 export const getRawLogsForBlockIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (blockHash: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number, logIndex9: number, logIndex10: number) => {
513   const response = await fetch(`.${MNEE_API}/v1/block/${blockHash}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}/${logIndex9}/${logIndex10}`);
514   if (!response.ok) {
515     throw new Error("Failed to get raw logs for block index index index index index index index index index index");
516   }
517   return response.json() as string[];
518 }
519
520 export const getRawLogsForRangeIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (startBlock: string, endBlock: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number, logIndex9: number, logIndex10: number) => {
521   const response = await fetch(`.${MNEE_API}/v1/blocks?start=${startBlock}&end=${endBlock}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}/${logIndex9}/${logIndex10}`);
522   if (!response.ok) {
523     throw new Error("Failed to get raw logs for range index index index index index index index index index index");
524   }
525   return response.json() as string[];
526 }
527
528 export const getRawLogForIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number, logIndex9: number, logIndex10: number, logIndex11: number) => {
529   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}/${logIndex9}/${logIndex10}/${logIndex11}`);
530   if (!response.ok) {
531     throw new Error("Failed to get raw log for index index");
532   }
533   return response.json() as string;
534 }
535
536 export const getRawLogsForTxIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number, logIndex9: number, logIndex10: number, logIndex11: number) => {
537   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}/${logIndex9}/${logIndex10}/${logIndex11}`);
538   if (!response.ok) {
539     throw new Error("Failed to get raw logs for tx index index");
540   }
541   return response.json() as string[];
542 }
543
544 export const getRawLogsForBlockIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (blockHash: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number, logIndex9: number, logIndex10: number, logIndex11: number) => {
545   const response = await fetch(`.${MNEE_API}/v1/block/${blockHash}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}/${logIndex9}/${logIndex10}/${logIndex11}`);
546   if (!response.ok) {
547     throw new Error("Failed to get raw logs for block index index");
548   }
549   return response.json() as string[];
550 }
551
552 export const getRawLogsForRangeIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (startBlock: string, endBlock: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number, logIndex9: number, logIndex10: number, logIndex11: number) => {
553   const response = await fetch(`.${MNEE_API}/v1/blocks?start=${startBlock}&end=${endBlock}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}/${logIndex9}/${logIndex10}/${logIndex11}`);
554   if (!response.ok) {
555     throw new Error("Failed to get raw logs for range index index");
556   }
557   return response.json() as string[];
558 }
559
560 export const getRawLogForIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number, logIndex9: number, logIndex10: number, logIndex11: number, logIndex12: number) => {
561   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}/${logIndex9}/${logIndex10}/${logIndex11}/${logIndex12}`);
562   if (!response.ok) {
563     throw new Error("Failed to get raw log for index index");
564   }
565   return response.json() as string;
566 }
567
568 export const getRawLogsForTxIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (txid: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number, logIndex9: number, logIndex10: number, logIndex11: number, logIndex12: number) => {
569   const response = await fetch(`.${MNEE_API}/v1/tx/${txid}/raw/logs/${logIndex}/${logIndex2}/${logIndex3}/${logIndex4}/${logIndex5}/${logIndex6}/${logIndex7}/${logIndex8}/${logIndex9}/${logIndex10}/${logIndex11}/${logIndex12}`);
570   if (!response.ok) {
571     throw new Error("Failed to get raw logs for tx index index");
572   }
573   return response.json() as string[];
574 }
575
576 export const getRawLogsForBlockIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndexIndex = async (blockHash: string, logIndex: number, logIndex2: number, logIndex3: number, logIndex4: number, logIndex5: number, logIndex6: number, logIndex7: number, logIndex8: number, logIndex9: number, logIndex1
```



```
mnee-portal-master > src > lib > ts authOptions.ts > authOptions > callbacks > jwt
  7  export const authOptions: NextAuthOptions = {
  8    providers: [
  9      async authorize(credentials) {
 10        ...
 11        return {
 12          id: user.id,
 13          name: user.name,
 14          email: user.email,
 15          image: user.image,
 16          emailVerified: user.emailVerified,
 17          requiresPasswordReset: user.requiresPasswordReset,
 18        };
 19      },
 20    ],
 21  },
 22  session: {
 23    strategy: 'jwt',
 24    maxAge: 24 * 60 * 60, // 24 hours
 25  },
 26  pages: {
 27    signIn: '/login',
 28    signOut: '/logout',
 29    error: '/login',
 30  },
 31  callbacks: {
 32    async jwt({ token, user, trigger }) {
 33      if (user) {
 34        // Initial sign in
 35        token.id = user.id;
 36        token.requiresPasswordReset = user.requiresPasswordReset;
 37      }
 38
 39      // On token update, refresh user data
 40      if (trigger === "update") {
 41        const freshUser = await prisma.user.findUnique({
 42          where: { id: token.id as string },
 43          select: { requiresPasswordReset: true }
 44        });
 45
 46        if (freshUser) {
 47          token.requiresPasswordReset = freshUser.requiresPasswordReset;
 48        }
 49      }
 50    }
 51  }
 52}
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
```

Broken Access Control

Resolved

Description

The application lacks controls for access control possibly allowing -

- a. Any user to approve a mint request by providing a userId.
- b. Any user can delete a mint request by providing Mint request id. Additionally, any user can delete a mint request of himself as well as of any other user.
- c. Identify pending requests
- d. List Approvers
- e. Set password resets for any email address by providing a valid email address

Vulnerable File

mnee-portal-master/scripts/approve.ts
mnee-portal-master/scripts/delete-mint-request.ts
mnee-portal-master/scripts/find-pending.ts
mnee-portal-master/scripts/list-approvers.ts

Impact

Sybil Attacks – A malicious actor can create multiple fake accounts to approve their own requests.

Collusion – Groups can approve each other's mints to manipulate token supply.

Spamming – Random users could approve unrelated mint requests, cluttering the approval system.

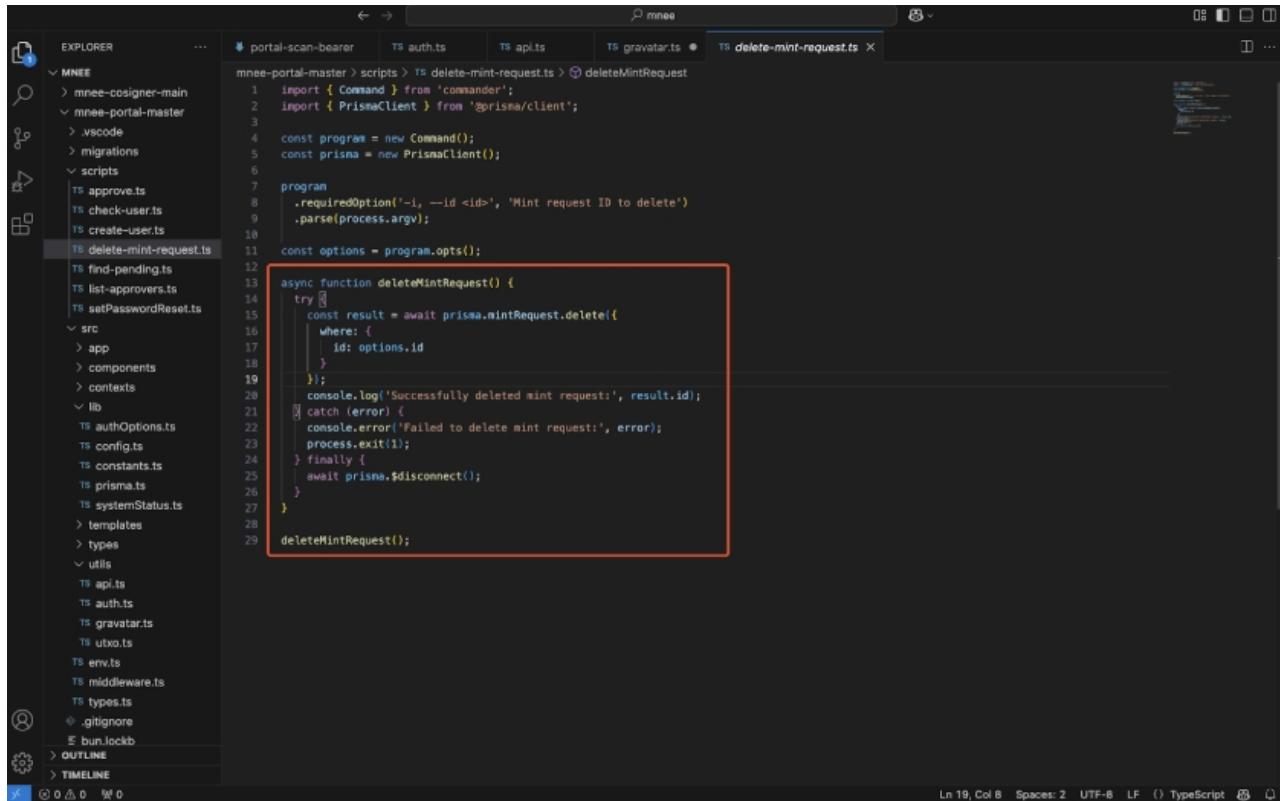
Attackers can delete any mint request, causing financial and operational disruptions.

Gain unauthorised access to data

Recommendation

Before approving, check if userId has approval rights

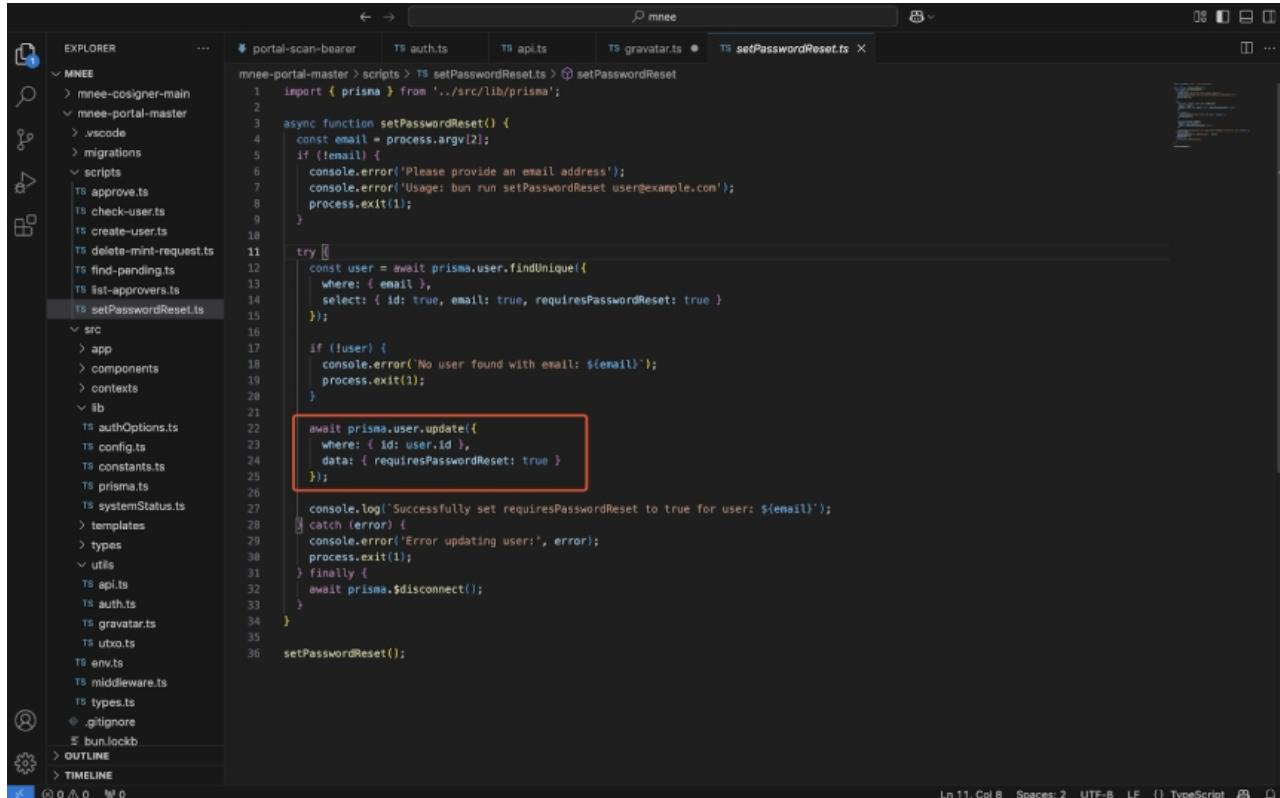
POC



```

mnee-portal-master > scripts > TS delete-mint-request.ts > deleteMintRequest
1 import { Command } from 'commander';
2 import { PrismaClient } from 'gprisma/client';
3
4 const program = new Command();
5 const prisma = new PrismaClient();
6
7 program
8 .requiredOption('-i, --id <id>', 'Mint request ID to delete')
9 .parse(process.argv);
10
11 const options = program.opts();
12
13 async function deleteMintRequest() {
14   try {
15     const result = await prisma.mintRequest.delete({
16       where: {
17         id: options.id
18       }
19     });
20     console.log('Successfully deleted mint request:', result.id);
21   } catch (error) {
22     console.error('Failed to delete mint request:', error);
23     process.exit(1);
24   } finally {
25     await prisma.$disconnect();
26   }
27 }
28
29 deleteMintRequest();

```



```

mnee-portal-master > scripts > TS setPasswordReset.ts > setPasswordReset
1 import { prisma } from '../src/lib/prisma';
2
3 async function setPasswordReset() {
4   const email = process.argv[2];
5   if (!email) {
6     console.error('Please provide an email address');
7     console.error('Usage: bun run setPasswordReset user@example.com');
8     process.exit(1);
9   }
10
11   try {
12     const user = await prisma.user.findUnique({
13       where: { email },
14       select: { id: true, email: true, requiresPasswordReset: true }
15     });
16
17     if (!user) {
18       console.error(`No user found with email: ${email}`);
19       process.exit(1);
20     }
21
22     await prisma.user.update({
23       where: { id: user.id },
24       data: { requiresPasswordReset: true }
25     });
26
27     console.log(`Successfully set requiresPasswordReset to true for user: ${email}`);
28   } catch (error) {
29     console.error('Error updating user:', error);
30     process.exit(1);
31   } finally {
32     await prisma.$disconnect();
33   }
34 }
35
36 setPasswordReset();

```

Broken Access Control

Acknowledged

Description

The code is not adding the session token or authentication cookie in the fetch request to MNEE_API/v1/transfer.

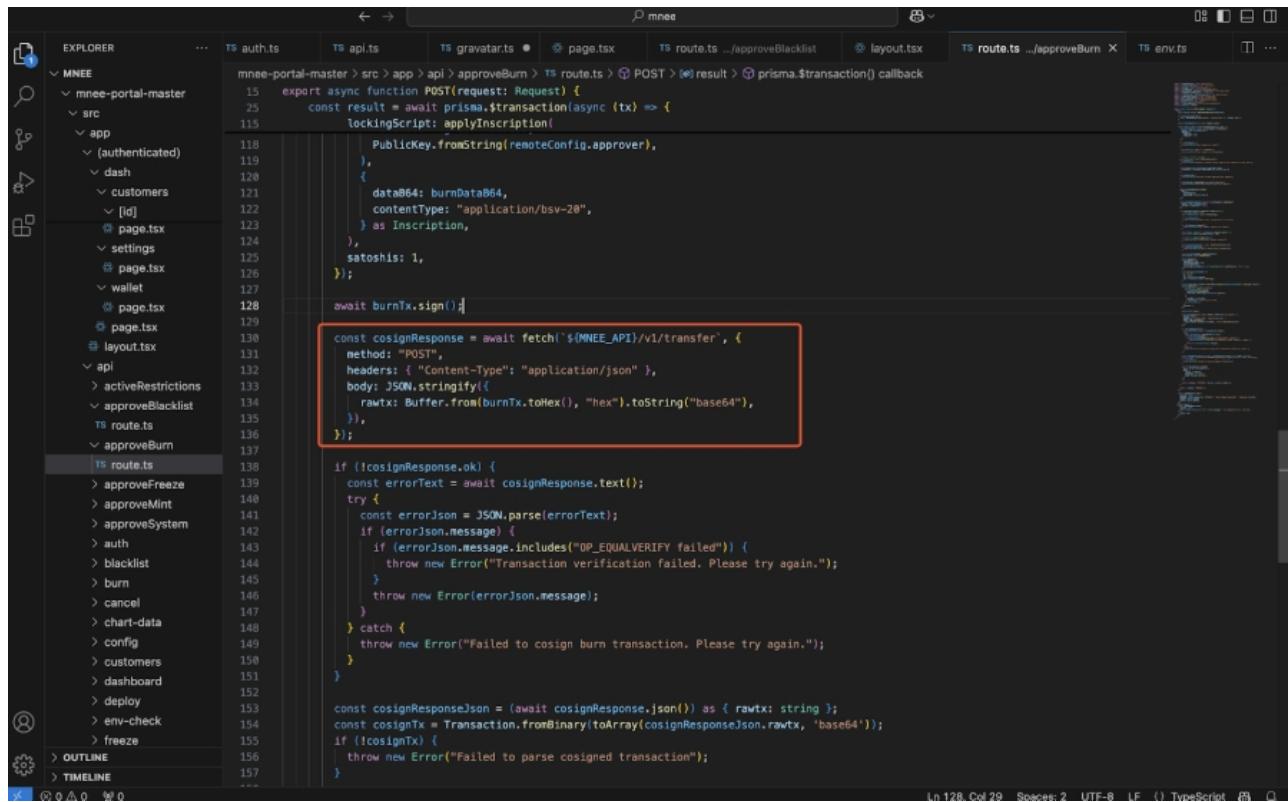
Vulnerable File

mnee-portal-master/src/app/api/approveBurn/route.ts
mnee-portal-master/src/app/api/approveMint/route.ts
mnee-portal-master/src/app/api/deploy/route.ts
mnee-portal-master/src getContexts/CustomerContext.tsx

Impact

The request to MNEE_API/v1/transfer lacks authentication, potentially allowing unauthorized transactions or exposing sensitive operations to attackers.

This issue is observed application wide and should be fixed at all vulnerable instances.



```

mnee-portal-master > src > app > api > approveBurn > route.ts > POST > result > prisma.$transaction() callback
15 export async function POST(request: Request) {
25   const result = await prisma.$transaction(async (tx) => {
115     lockingScript: applyInscription(
       ),
       {
         data64: burnData64,
         contentType: "application/bsv-28",
       } as Inscription,
       ),
       satoshis: 1,
     });
128   await burnTx.sign();
130
const cosignResponse = await fetch(`${MNEE_API}/v1/transfer`, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({
    rawtx: Buffer.from(burnTx.toHex()), "hex").toString("base64"),
  }),
});
137
if (!cosignResponse.ok) {
  const errorText = await cosignResponse.text();
  try {
    const errorJson = JSON.parse(errorText);
    if (errorJson.message) {
      if (errorJson.message.includes("OP_EQUALVERIFY failed")) {
        throw new Error("Transaction verification failed. Please try again.");
      }
      throw new Error(errorJson.message);
    }
  } catch {
    throw new Error("Failed to cosign burn transaction. Please try again.");
  }
}
148
const cosignResponseJson = (await cosignResponse.json()) as { rawtx: string };
const cosignTx = Transaction.fromBinary(toArray(cosignResponseJson.rawtx, 'base64'));
if (!cosignTx) {
  throw new Error("Failed to parse cosigned transaction");
}

```

Amount Value is Not Validated During Burn

Resolved

Description

It is possible that an attacker can supply a negative value for the amount while trying to execute a Burn request. The code currently does not check if the value of the amount parameter is greater than 0.

Vulnerable File

mnee-portal-master/src/app/api/burn/route.ts

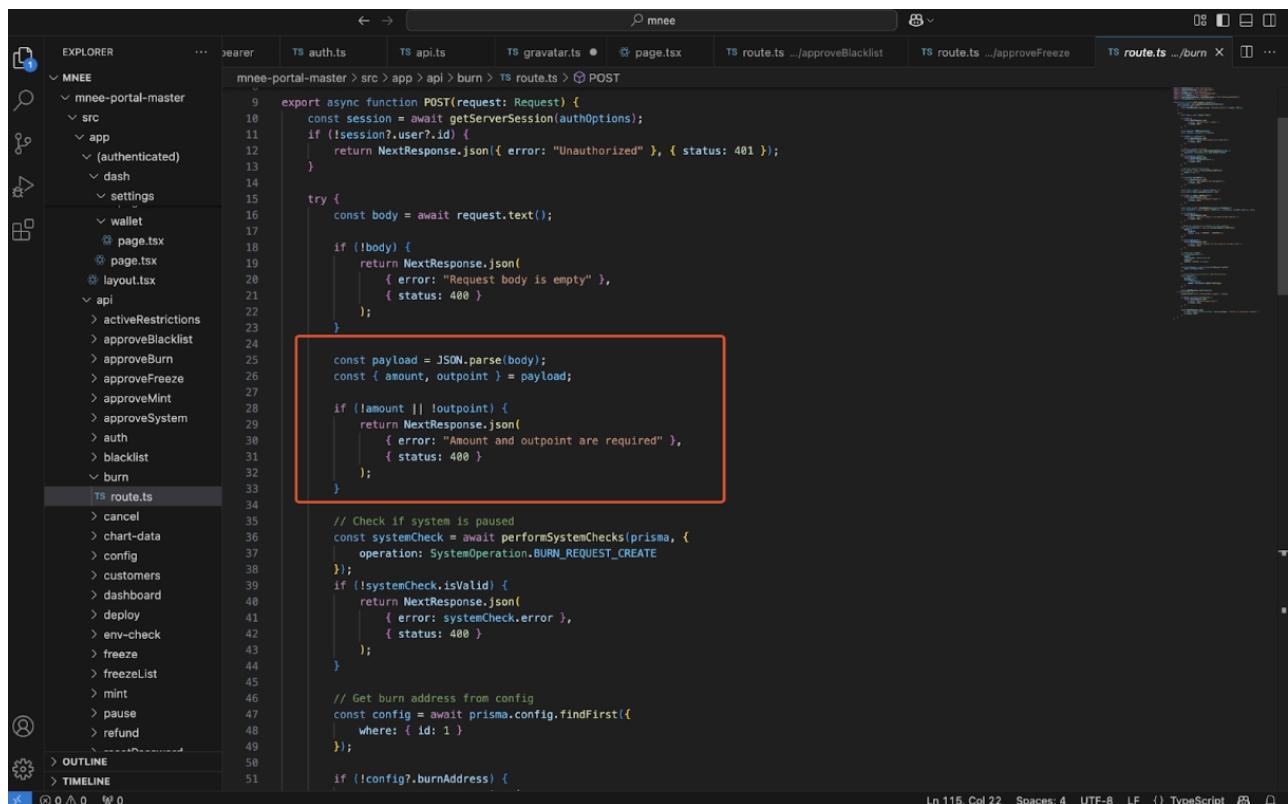
Impact

Attackers can submit a negative value or 0 as an amount. This may cause unintended behaviour and possible loss of funds.

Recommendation

Implement checks for the value of the supplied amount to be greater than 0.

Additionally the code should also check if the user trying to execute the request is blacklisted or not.



```

    export async function POST(request: Request) {
        const session = await getServerSession(authOptions);
        if (!session?.user?.id) {
            return NextResponse.json({ error: "Unauthorized" }, { status: 401 });
        }

        try {
            const body = await request.text();

            if (!body) {
                return NextResponse.json(
                    { error: "Request body is empty" },
                    { status: 400 }
                );
            }

            const payload = JSON.parse(body);
            const { amount, outpoint } = payload;

            if (!amount || !outpoint) {
                return NextResponse.json(
                    { error: "Amount and outpoint are required" },
                    { status: 400 }
                );
            }

            // Check if system is paused
            const systemCheck = await performSystemChecks(prisma, {
                operation: SystemOperation.BURN_REQUEST_CREATE
            });
            if (!systemCheck.isValid) {
                return NextResponse.json(
                    { error: systemCheck.error },
                    { status: 400 }
                );
            }

            // Get burn address from config
            const config = await prisma.config.findFirst({
                where: { id: 1 }
            });

            if (!config?.burnAddress) {
                ...
            }
        }
    }
}

```

Unvalidated Blacklisted Request Cancellation

Resolved

Description

Unlike other request types, blacklistRequestId should check if session.user.id matches requestedBy. If a match is found, the request should be denied. Currently, the code is missing this match.

Vulnerable File

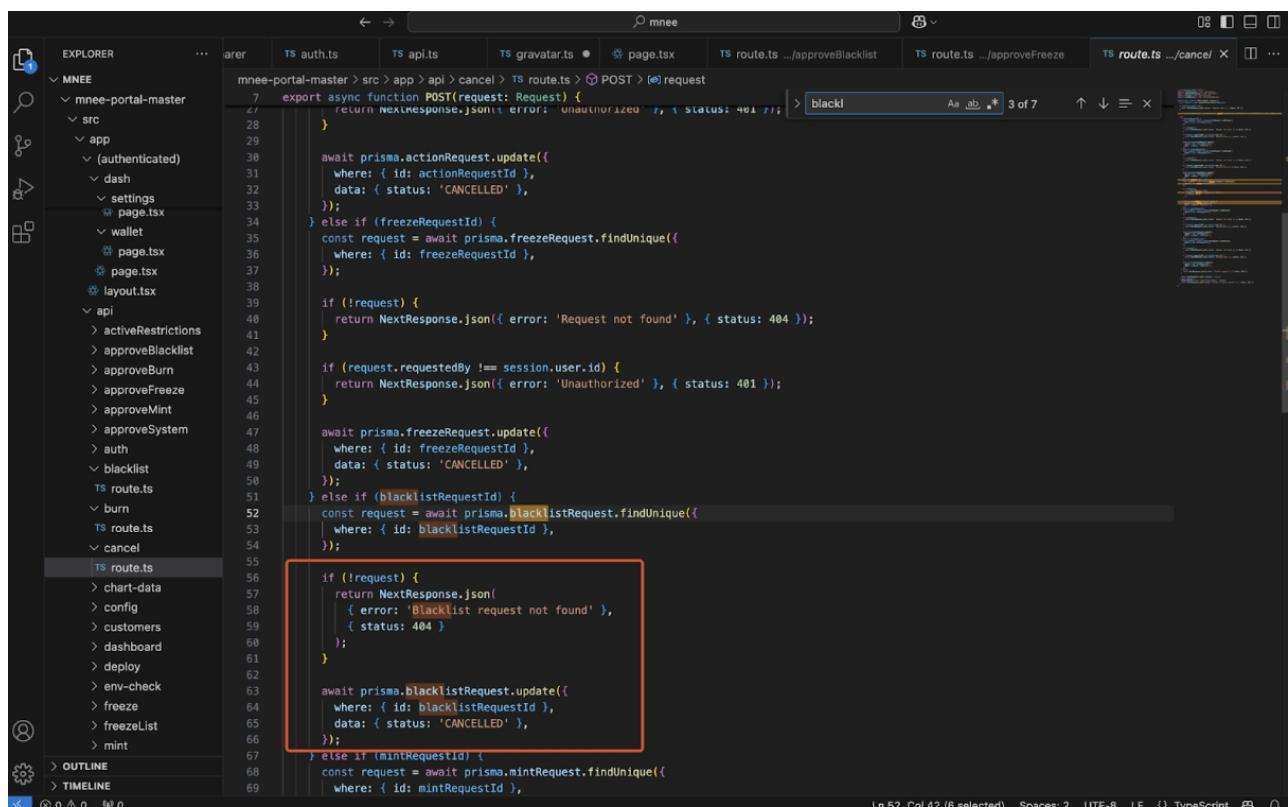
mnee-portal-master/src/app/api/cancel/route.ts

Impact

This might allow the same user who is blacklisted to cancel a request.

Recommendation

Implement a check if session.user.id matches requestedBy. If a match is found, the request should be denied. Currently, the code is missing this match.



```

    export async function POST(request: Request) {
        return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
    }

    await prisma.actionRequest.update({
        where: { id: actionRequestId },
        data: { status: 'CANCELLED' },
    });

    else if (freezeRequestId) {
        const request = await prisma.freezeRequest.findUnique({
            where: { id: freezeRequestId },
        });

        if (!request) {
            return NextResponse.json({ error: 'Request not found' }, { status: 404 });
        }

        if (request.requestedBy !== session.user.id) {
            return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
        }

        await prisma.freezeRequest.update({
            where: { id: freezeRequestId },
            data: { status: 'CANCELLED' },
        });

    } else if (blacklistRequestId) {
        const request = await prisma.blacklistRequest.findUnique({
            where: { id: blacklistRequestId },
        });

        if (!request) {
            return NextResponse.json({
                error: 'Blacklist request not found',
                status: 404
            });
        }

        await prisma.blacklistRequest.update({
            where: { id: blacklistRequestId },
            data: { status: 'CANCELLED' },
        });

    } else if (mintRequestId) {
        const request = await prisma.mintRequest.findUnique({
            where: { id: mintRequestId },
        });
    }
}

```

Broken Access Control

Resolved

Description

The POST and DELETE routes allow updates and deletions of the configuration without authentication. This means anyone who has access to the API could modify or delete critical configuration settings, including the minting and burning process.

Vulnerable File

mnee-portal-master/src/app/api/config/route.ts

Impact

Anyone who has access to the API could modify or delete critical configuration settings.

```

import { NextResponse } from "next/server";
import prismadb from "@/lib/prismadb";
import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

export async function POST(request: Request) {
  const { tokenid, feeAddress, decimals, latestMinterTx } = await request.json();
  const mintAddress = PrivateKey.fromString("MINT_WIF").toAddress();
  const burnAddress = PrivateKey.fromString("BURN_WIF").toAddress();
  try {
    // Get current config to keep existing fees
    const currentConfig = await prisma.config.findUnique({
      where: { id: 1 },
    });
    const config = await prisma.config.upsert({
      where: { id: 1 },
      update: {
        tokenid,
        feeAddress,
        decimals,
        latestMinterTx,
        mintAddress,
        burnAddress,
        // Keep existing fees or use default
        fees: currentConfig.fees ?? defaultFees,
      },
      create: {
        id: 1,
        tokenid,
        feeAddress,
        decimals,
        latestMinterTx,
        fundAddress: "",
        mintAddress: "",
        burnAddress: "",
        // Use default fees for new config
        fees: defaultFees,
      },
    });
    // Revalidate cache after update
    await revalidateConfig();
    return NextResponse.json(config);
  } catch (error) {
    console.error("Error saving config:", error);
    return NextResponse.json({
      errors: ["Error saving configuration."],
      status: 500,
    });
  }
}

export async function DELETE() {
  try {
    await prisma.config.deleteMany();
    return NextResponse.json({
      message: "Configuration cleared.",
      headers: { "Cache-Control": "no-store" },
    });
  } catch (error) {
    ...
  }
}

```

Unauthorised Refund

Resolved

Description

The refund logic in the POST request handler allows users to specify any refundAddress, without verifying whether the address belongs to the original sender. This introduces a security vulnerability, as attackers can exploit this mechanism to redirect refunds to their own wallets, stealing funds from legitimate users.

Vulnerable File

mnee-portal-master/src/app/api/refund/route.ts

Impact

Attackers can request refunds for someone else's UTXO but provide their own address, effectively stealing the funds.

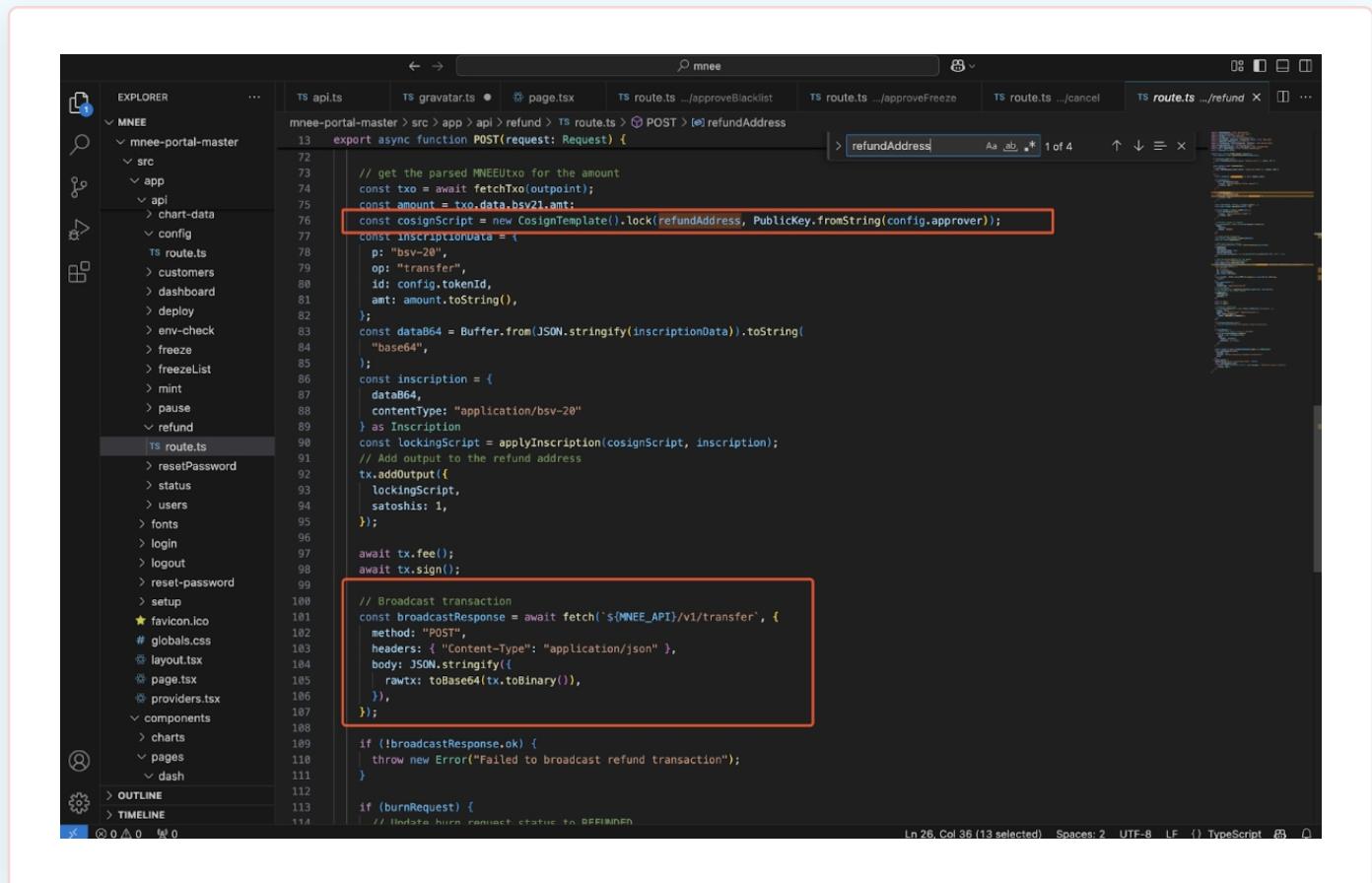
Recommendation

The refund address should be fetched from the original transaction sender, not from user input.

POC

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists project files and components. The main area displays a TypeScript file named `ts route.ts`. A red box highlights a section of the code that checks for required fields in the request body: `outpoint` and `refundAddress`. If either is missing, it returns a 400 Bad Request response with an error message. The code then extracts the `vout` from the `outpoint` string and checks if it's a valid number. If not, it returns a 400 Bad Request response with an error message. Finally, it uses Prisma to find a burn request by the `vout`.

```
mnee-portal-master > src > app > api > refund > ts routes > POST > refundAddress
12
13 export async function POST(request: Request) {
14   const session = await getServerSession(authOptions);
15
16   if (!session?.user?.id) {
17     return NextResponse.json({ error: "Unauthorized" }, { status: 401 });
18   }
19
20   const config = await fetchConfig();
21   if (!config) {
22     return NextResponse.json({ error: "Config not found" }, { status: 404 });
23   }
24
25   try {
26     const { outpoint, refundAddress } = await request.json();
27
28     if (!outpoint) {
29       return NextResponse.json(
30         { error: "Missing required field: outpoint" },
31         { status: 400 }
32     );
33   }
34
35   if (!refundAddress) {
36     return NextResponse.json(
37       { error: "Missing required field: refundAddress" },
38       { status: 400 }
39     );
40   }
41
42   const [sourceTXID, voutStr] = outpoint.split('_');
43   const vout = Number.parseInt(voutStr, 10);
44
45   if (!sourceTXID || Number.isNaN(vout)) {
46     return NextResponse.json(
47       { error: "Invalid outpoint format" },
48       { status: 400 }
49     );
50   }
51
52   // Find burn request by outpoint
53   const burnRequest = await prisma.burnRequest.findFirst({
54     where: {
```



```

mnee-portal-master > src > app > api > refund > TS route.ts > POST > [e] refundAddress
13  export async function POST(request: Request) {
14    // get the parsed MNEEUtxo for the amount
15    const txo = await fetchTxo(outputInt);
16    const amount = txo.data.bsv21 amt;
17    const cosignScript = new CosignTemplate().lock(refundAddress, PublicKey.fromString(config.approver));
18    const inscriptionData = {
19      p: "bsv-20",
20      op: "transfer",
21      id: config tokenId,
22      amt: amount.toString(),
23    };
24    const data64 = Buffer.from(JSON.stringify(inscriptionData)).toString(
25      "base64",
26    );
27    const inscription = {
28      data64,
29      contentType: "application/bbv-20"
30    } as Inscription;
31    const lockingScript = applyInscription(cosignScript, inscription);
32    // Add output to the refund address
33    tx.addOutput({
34      lockingScript,
35      satoshis: 1,
36    });
37
38    await tx.fee();
39    await tx.sign();
40
41    // Broadcast transaction
42    const broadcastResponse = await fetch(`${MNEE_API}/v1/transfer`, {
43      method: "POST",
44      headers: { "Content-Type": "application/json" },
45      body: JSON.stringify({
46        rawtx: toBase64(tx.toBinary()),
47      }),
48    });
49
50    if (!broadcastResponse.ok) {
51      throw new Error("Failed to broadcast refund transaction");
52    }
53
54    if (burnRequest) {
55      // Update burn request status to REFINED
56    }
57  }

```



Medium Severity Issues

Usage of weak hashing library (MD5)

Resolved

Description

Using a weak hashing library like MD5 increases the risk of data breaches. MD5 is vulnerable to collision attacks, where two different inputs produce the same output, compromising data integrity and security.

Vulnerable File

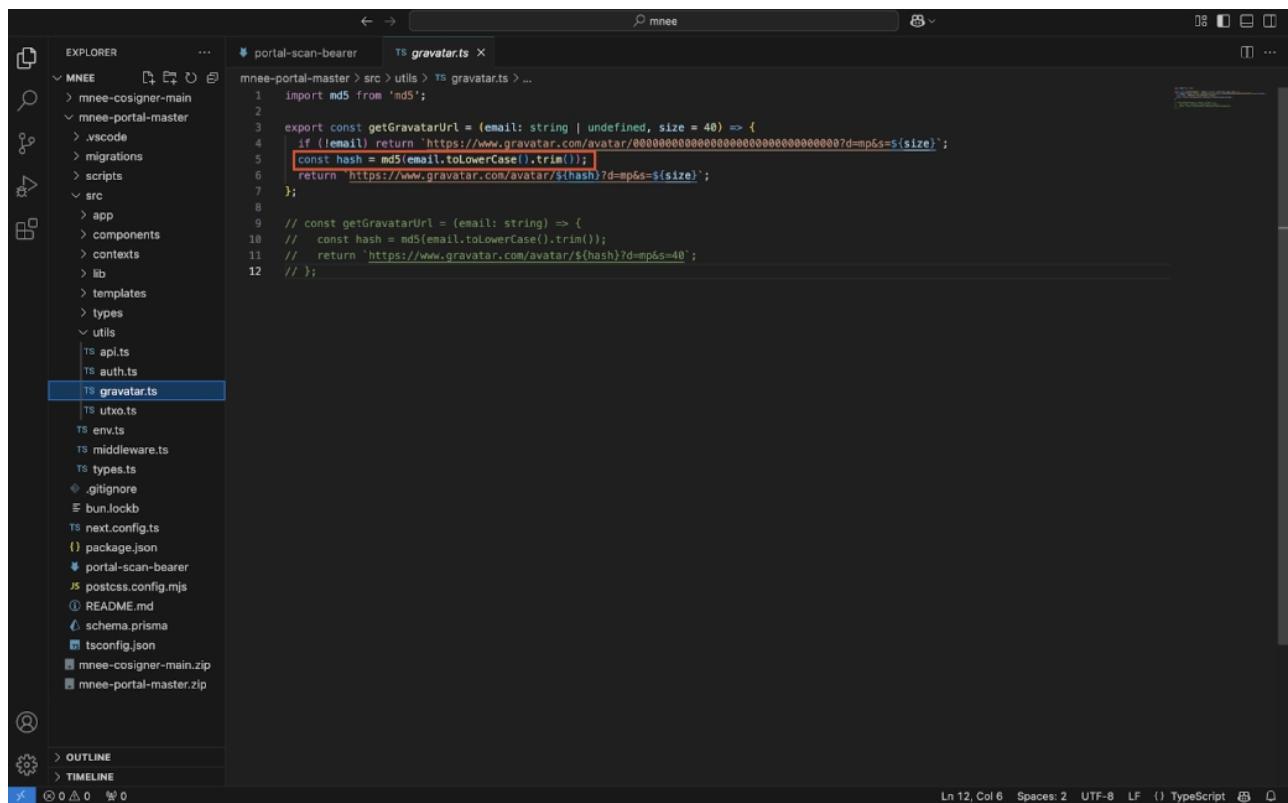
mnee-portal/src/utils/gravatar.ts:5

Impact

1. MD5 is vulnerable to collision attacks, where two different inputs produce the same output, compromising data integrity and security.

Recommendation

- Do not use MD5 for hashing or security purposes. It is no longer considered secure for cryptographic applications.
 - Do opt for stronger hashing algorithms like SHA-256 for enhanced security.
- ```
const crypto = require("crypto");
const key = "secret key";
const hash = crypto.createHmac("sha256",key).update(user.password).digest('hex');
```



```
import md5 from 'md5';
export const getGravatarUrl = (email: string | undefined, size = 40) => {
 if (!email) return `https://www.gravatar.com/avatar/00000000000000000000000000000000?d=mp&s=${size}`;
 const hash = md5(email.toLowerCase().trim());
 return `https://www.gravatar.com/avatar/${hash}?d=mp&s=${size}`;
};
```



## Insecure Input Validation

Resolved

### Description

The application does not validate and sanitize input values. User input should always be validated and sanitized before processing through APIs or functions. The txid is directly fetched from the URL and processed in the function without verification if it is a valid txid.

### Vulnerable File

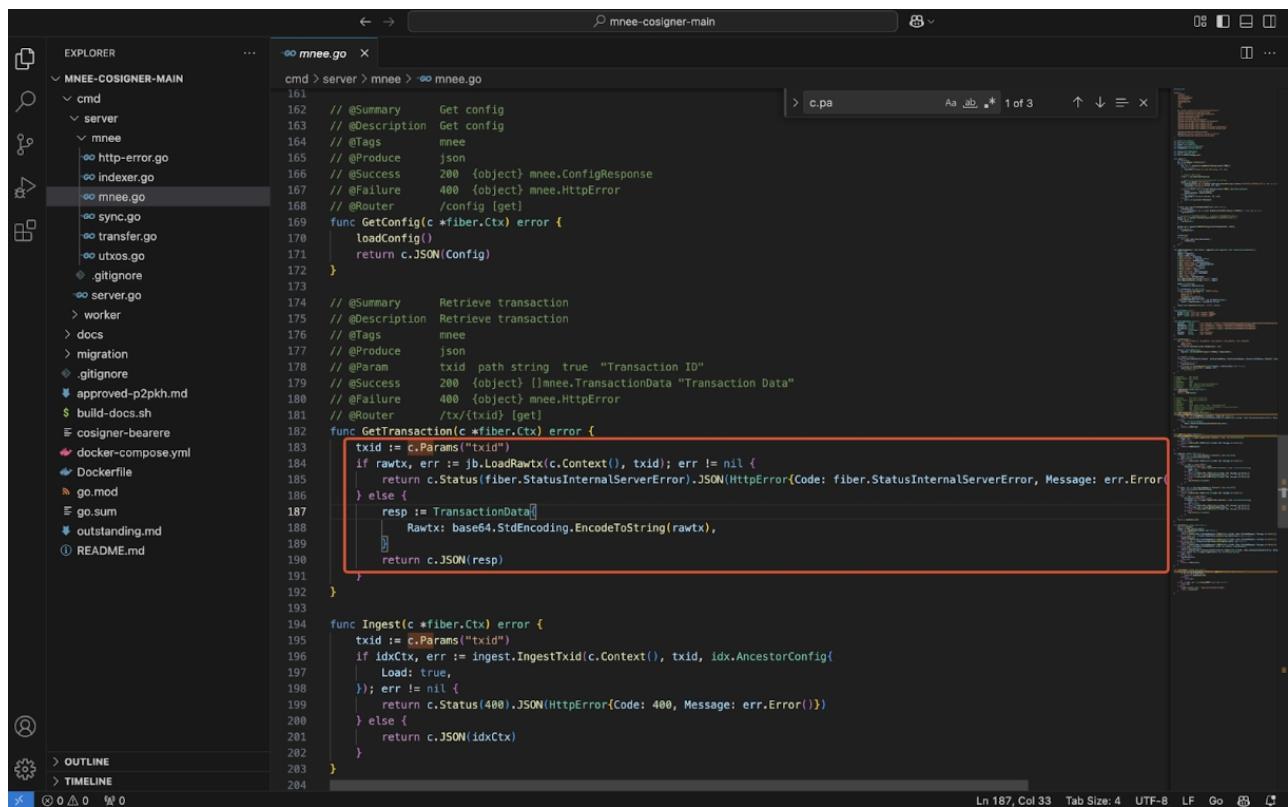
mnee-cosigner/cmd/server/mnee/mnee.go

### Impact

It is possible to inject malicious values in the API leading to unexpected behaviour in the application.

### Recommendation

Implement a regex of function that checks if input parsed is valid for the specific function and no insecure or malicious value is being parsed.



```

161 // @Summary Get config
162 // @Description Get config
163 // @Tags mnee
164 // @Produce json
165 // @Success 200 {object} mnee.ConfigResponse
166 // @Failure 400 {object} mnee.HttpError
167 // @Router /config {get}
168 func GetConfig(c *fiber.Ctx) error {
169 loadConfig()
170 return c.JSON(Config)
171 }
172
173 // @Summary Retrieve transaction
174 // @Description Retrieve transaction
175 // @Tags mnee
176 // @Produce json
177 // @Param txid path string true "Transaction ID"
178 // @Success 200 {object} [lnee.TransactionData "Transaction Data"
179 // @Failure 400 {object} mnee.HttpError
180 // @Router /tx/{txid} {get}
181 func GetTransaction(c *fiber.Ctx) error {
182 txid := c.Params("txid")
183 if rawtx, err := jb.LoadRawTx(c.Context(), txid); err != nil {
184 return c.Status(fiber.StatusInternalServerError).JSON(err)
185 } else {
186 resp := TransactionData{
187 Rawtx: base64.StdEncoding.EncodeToString(rawtx),
188 }
189 return c.JSON(resp)
190 }
191 }
192
193 func Ingest(c *fiber.Ctx) error {
194 txid := c.Params("txid")
195 if idxCtx, err := ingest.IngestTxid(c.Context(), txid, idx.AncestorConfig{
196 Load: true,
197 }); err != nil {
198 return c.Status(400).JSON(err)
199 } else {
200 return c.JSON(idxCtx)
201 }
202 }
203
204

```

# No Rate Limit

## Acknowledged

## Description

The application has not implemented a rate limiter to limit users from executing multiple malicious requests on the application server leading to increased bandwidth and server load. All API functions should have a rate limiter defined to limit max 10 requests per minute as a normal user

## Vulnerable File

mnee-cosigner/cmd/server/mnee/mnee.go

## Impact

Attackers can send bulk data to API Endpoints. Due to lack of limitation, APIs would process this data and at one stage, the exhaustive pressure on the server will increase and very high bandwidth will be consumed leading to a DOS attack.

## Recommendation

Implement a rate limiter to limit max up to 10 requests per minute.

The screenshot shows a Go code editor interface with the following details:

- Explorer View:** Shows the project structure under "MNEE-COSIGNER-MAIN". The file "mnee.go" is currently selected.
- Search Bar:** Displays the search term "mnee-cosigner-main".
- Code Editor:** The main pane displays the content of "mnee.go". The code includes functions for handling transactions, ingestions, and digestions, utilizing the fiber framework and a custom idxCtx struct.
- Right Panel:** A large panel on the right side shows the full codebase of the project, with various files like "sync.go", "server.go", and "cmd.go" visible.

```
cmd > server > mnee > mnee.go
179 / @Success 200 {object} []mnee.TransactionData "Transaction Data"
180 / @Failure 400 {object} mnee.HttpError
181 / @Router /tx/{txid} [get]
182 func GetTransaction(c *fiber.Ctx) error {
183 txid := c.Params("txid")
184 if rawtx, err := jb.LoadRawtx(c.Context(), txid); err != nil {
185 return c.Status(fiber.StatusInternalServerError).JSON(&HttpError{Code: fiber.StatusInternalServerError, Message: err.Error()})
186 } else {
187 resp := TransactionData{
188 Rawtx: base64.StdEncoding.EncodeToString(rawtx),
189 }
190 return c.JSON(resp)
191 }
192 }
193
194 func Ingest(c *fiber.Ctx) error {
195 txid := c.Params("txid")
196 if idxCtx, err := ingest.IngestTxid(c.Context(), txid, idx.AncestorConfig{
197 Load: true,
198 }); err != nil {
199 return c.Status(400).JSON(&HttpError{Code: 400, Message: err.Error()})
200 } else {
201 return c.JSON(idxCtx)
202 }
203 }
204
205 func Digest(c *fiber.Ctx) error {
206 if txids, err := store.SearchMembers(c.Context(), &idx.SearchCfg{
207 Keys: []string{idx.PendingTxLog},
208 }); err != nil {
209 return c.Status(500).JSON(&HttpError{Code: 500, Message: err.Error()})
210 } else {
211 for _, txid := range txids {
212 log.Println("Digesting", txid)
213 if idxCtx, err := ingest.IngestTxid(c.Context(), txid, idx.AncestorConfig{
214 Load: true,
215 }); err != nil {
216 return c.Status(500).JSON(&HttpError{Code: 500, Message: err.Error()})
217 } else if out, err := json.MarshalIndent(idxCtx, "", " "); err != nil {
218 return c.Status(500).JSON(&HttpError{Code: 500, Message: err.Error()})
219 } else {
220 log.Println(string(out))
221 }
222 }
223 }
224 }
```

## MNEE Team's Comment

Rate Limit is handled through our proxy server for now.



# Low Severity Issues

## Leakage of sensitive information in logger message

Resolved

### Description

The application logs sensitive information unwantedly in the user's browser. Sensitive information leakage through logger messages can compromise user privacy and security. This vulnerability occurs when sensitive data, such as personal identifiable information (PII), is included in log messages, making it accessible to unauthorized individuals.

### Vulnerable File

mnee-portal-master/

```
src/app/api/resetPassword/route.ts:18
scripts/check-user.ts:23
scripts/create-user.ts:37
scripts/find-pending.ts:25
scripts/list-approvers.ts:49
scripts/list-approvers.ts:52
scripts/list-approvers.ts:61
src/app/api/approveBlacklist/route.ts:50
src/app/api/approveFreeze/route.ts:44
src/app/api/approveFreeze/route.ts:106
src/app/api/approveFreeze/route.ts:141
src/app/api/resetPassword/route.ts:18
src/components/pages/dash/content/customers/view.tsx:82
```

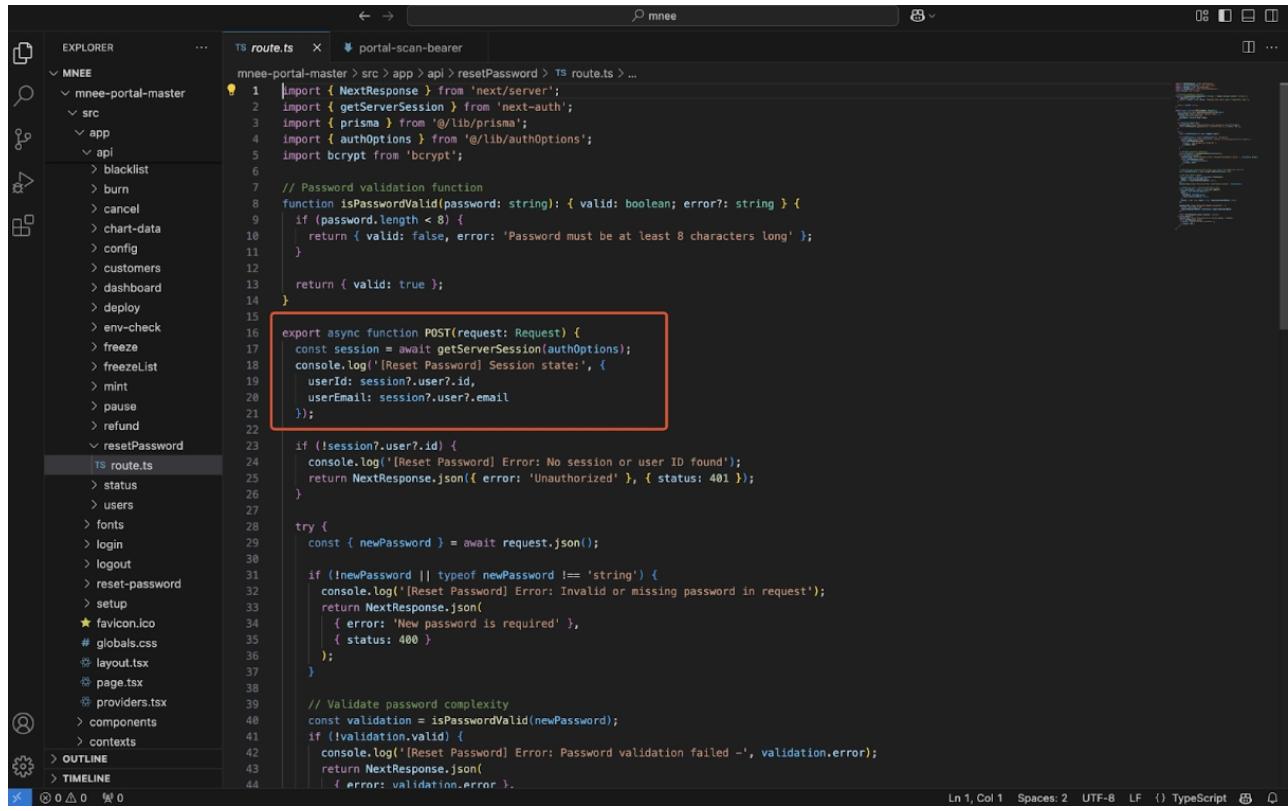
### Recommendation

Do not include sensitive data in logger messages. This can lead to unintended exposure of private information. Do use non-sensitive, unique identifiers to reference users in log messages. This approach maintains user privacy while still allowing for effective logging.  
logger.info('User is: \${user.uuid}')

### Impact

Logging sensitive information can allow attackers to steal logged information under certain conditions.

## POC



```
1 import { NextResponse } from 'next/server';
2 import { getServerSession } from 'next-auth';
3 import { prisma } from '@lib/prisma';
4 import { authOptions } from '@lib/authOptions';
5 import bcrypt from 'bcrypt';
6
7 // Password validation function
8 function isPasswordValid(password: string): { valid: boolean; error?: string } {
9 if (password.length < 8) {
10 return { valid: false, error: 'Password must be at least 8 characters long' };
11 }
12
13 return { valid: true };
14 }
15
16 export async function POST(request: Request) {
17 const session = await getServerSession(authOptions);
18 console.log(`[Reset Password] Session state:`, {
19 userId: session?.user?.id,
20 userEmail: session?.user?.email
21 });
22
23 if (!session?.user?.id) {
24 console.log(`[Reset Password] Error: No session or user ID found`);
25 return NextResponse.json({ error: 'Unauthorized' }, { status: 401 });
26 }
27
28 try {
29 const { newPassword } = await request.json();
30
31 if (!newPassword || typeof newPassword !== 'string') {
32 console.log(`[Reset Password] Error: Invalid or missing password in request`);
33 return NextResponse.json(
34 { error: 'New password is required' },
35 { status: 400 }
36);
37 }
38
39 // Validate password complexity
40 const validation = isPasswordValid(newPassword);
41 if (!validation.valid) {
42 console.log(`[Reset Password] Error: Password validation failed - ${validation.error}`);
43 return NextResponse.json(
44 { error: validation.error }
45);
46 }
47 }
48 }
```

## Middleware Does not Handle Token Expiration

Resolved

### Description

The middleware only checks if a token exists (!!token), but doesn't verify if it is expired.

### Vulnerable File

mnee-portal-master/src/middleware.ts

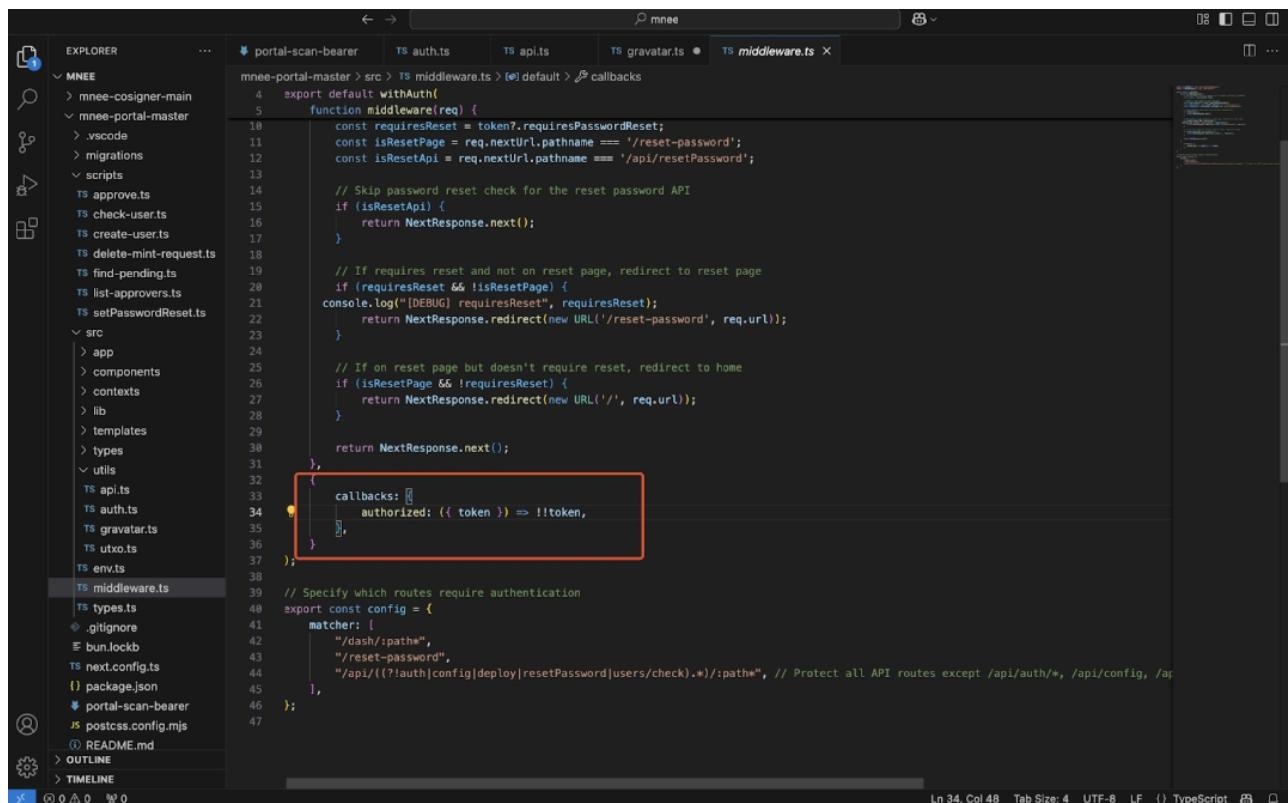
### Impact

It is possible to use old generated tokens if tokens are not expired. Old password reset tokens if re-used by hackers could lead to severe security issues.

### Recommendation

Modify the authorized callback to check token expiration

### POC



```

1 import { NextResponse } from "next";
2 import { withAuth } from "mnee-portal-main";
3
4 export default withAuth({
5 function middleware(req) {
6 const token = req.cookies["token"];
7 const requiresReset = token?.requiresPasswordReset;
8 const isLoginPage = req.nextUrl.pathname === "/reset-password";
9 const isResetApi = req.nextUrl.pathname === "/api/resetPassword";
10
11 // Skip password reset check for the reset password API
12 if (isResetApi) {
13 return NextResponse.next();
14 }
15
16 // If requires reset and not on reset page, redirect to reset page
17 if (requiresReset && !isLoginPage) {
18 console.log("[DEBUG] requiresReset", requiresReset);
19 return NextResponse.redirect(new URL('/reset-password', req.url));
20 }
21
22 // If on reset page but doesn't require reset, redirect to home
23 if (isLoginPage && !requiresReset) {
24 return NextResponse.redirect(new URL('/', req.url));
25 }
26
27 return NextResponse.next();
28 },
29
30 callbacks: [
31 {
32 authorized: ({ token }) => !!token,
33 },
34],
35 });
36
37
38 // Specify which routes require authentication
39 export const config = {
40 matcher: [
41 "/dash/:path",
42 "/reset-password",
43 "/api/((?!auth|config|deploy|resetPassword|users/check).*)/:path*", // Protect all API routes except /api/auth/*, /api/config, /ap
44],
45 },
46 };
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
307
308
309
309
310
311
312
313
313
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
```

## User Enumeration

Resolved

### Description

The application API prints generic error message if a supplied user email is not found in the database. Attackers can brute-force the email address field and identify valid registered email addresses.

### Vulnerable File

mnee-portal-master/scripts/check-user.ts

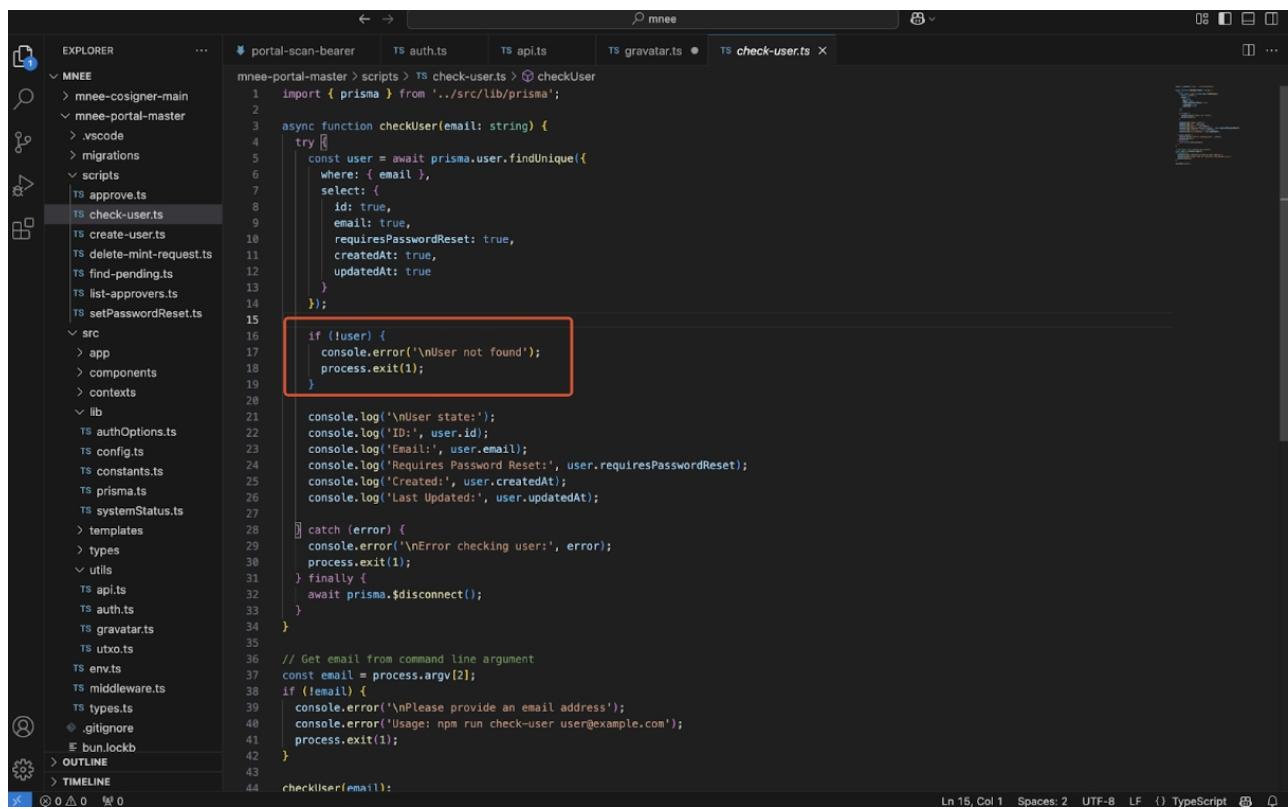
### Impact

Generic error messages can allow attackers to identify valid user email addresses. Using this information attackers could further plan their attack path against individual users.

### Recommendation

Print generic error message

### POC



```

import { prisma } from '../src/lib/prisma';

async function checkUser(email: string) {
 try {
 const user = await prisma.user.findUnique({
 where: { email },
 select: {
 id: true,
 email: true,
 requiresPasswordReset: true,
 createdAt: true,
 updatedAt: true
 }
 });
 if (!user) {
 console.error(`\nUser not found`);
 process.exit(1);
 }
 console.log(`\nUser state:`);
 console.log(`ID: ${user.id}`);
 console.log(`Email: ${user.email}`);
 console.log(`Requires Password Reset: ${user.requiresPasswordReset}`);
 console.log(`Created: ${user.createdAt}`);
 console.log(`Last Updated: ${user.updatedAt}`);
 } catch (error) {
 console.error(`\nError checking user: ${error}`);
 process.exit(1);
 } finally {
 await prisma.$disconnect();
 }
}

// Get email from command line argument
const email = process.argv[2];
if (!email) {
 console.error(`\nPlease provide an email address`);
 console.error(`Usage: npm run check-user user@example.com`);
 process.exit(1);
}
checkUser(email);

```

## Weak Password Check

Resolved

### Description

The application has a `isPasswordValid` function which only checks if password length is greater than 8 characters. It should also check for special characters, numbers, common password patterns, and numeric characters.

### Vulnerable File

`mnee-portal-master/scripts/create-user.ts`

`mnee-portal-master/src/app/reset-password/page.tsx`

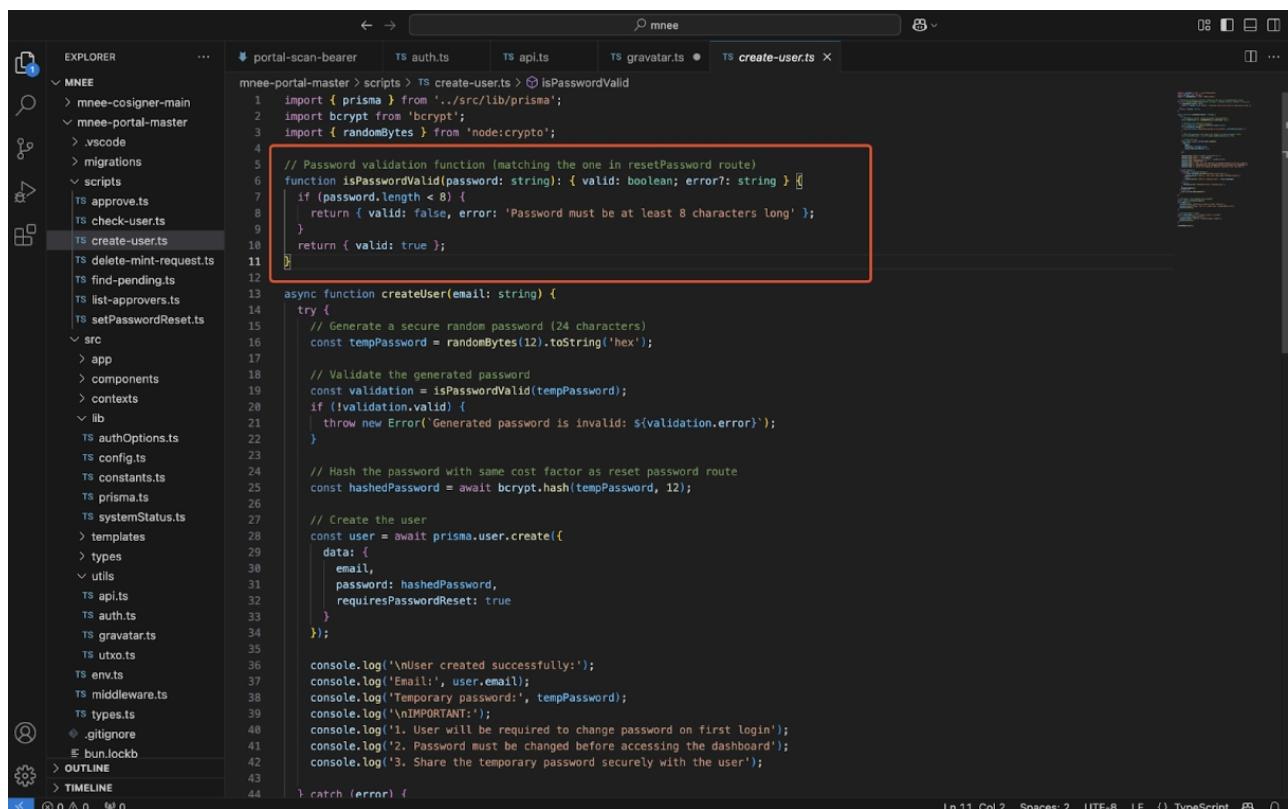
### Impact

Users might create 8 length passwords (example: `test@123` or `john@123`) but it would still be weak. It is important to implement checks for special characters, numbers, common password patterns, and numeric characters to create a strong password.

### Recommendation

Implement checks for special characters, numbers, common password patterns, and numeric characters

### POC



```

mnee-portal-master > scripts > ts create-user.ts > isPasswordValid
1 import { prisma } from '../src/lib/prisma';
2 import bcrypt from 'bcrypt';
3 import { randomBytes } from 'node:crypto';
4
5 // Password validation function (matching the one in resetPassword route)
6 function isPasswordValid(password: string): { valid: boolean; error?: string } {
7 if (password.length < 8) {
8 return { valid: false, error: 'Password must be at least 8 characters long' };
9 }
10 return { valid: true };
11}
12
13 async function createUser(email: string) {
14 try {
15 // Generate a secure random password (24 characters)
16 const tempPassword = randomBytes(12).toString('hex');
17
18 // Validate the generated password
19 const validation = isPasswordValid(tempPassword);
20 if (!validation.valid) {
21 throw new Error(`Generated password is invalid: ${validation.error}`);
22 }
23
24 // Hash the password with same cost factor as reset password route
25 const hashedPassword = await bcrypt.hash(tempPassword, 12);
26
27 // Create the user
28 const user = await prisma.user.create({
29 data: {
30 email,
31 password: hashedPassword,
32 requiresPasswordReset: true
33 }
34 });
35
36 console.log(`\nUser created successfully:`);
37 console.log(`Email: ${user.email}`);
38 console.log(`Temporary password: ${tempPassword}`);
39 console.log(`\nIMPORTANT:`);
40 console.log(`1. User will be required to change password on first login`);
41 console.log(`2. Password must be changed before accessing the dashboard`);
42 console.log(`3. Share the temporary password securely with the user`);
43
44 } catch (error) {

```

## Improper Error Handling

Resolved

### Description

The application currently logs API errors without sanitization. If an API request fails, the error message could expose internal details.

### Vulnerable File

mnee-portal-master/src/contexts/CustomerContext.tsx

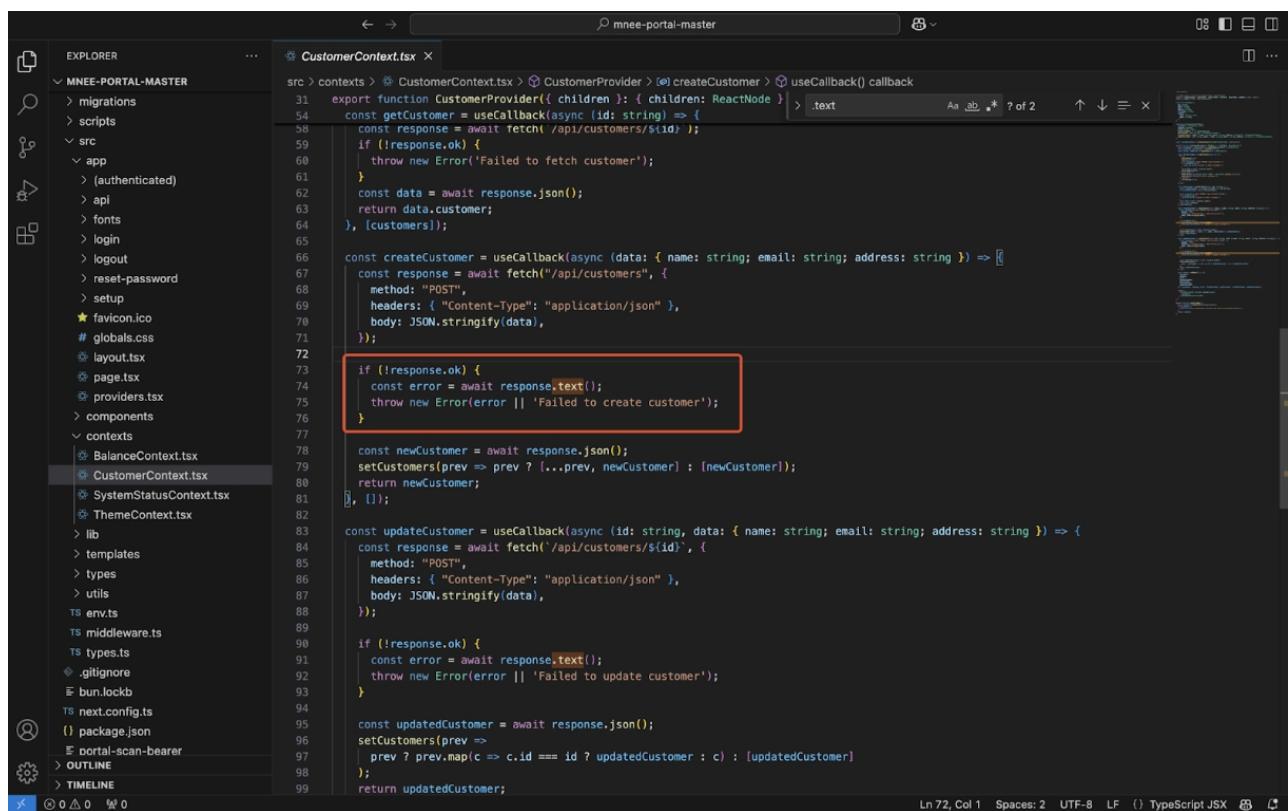
### Impact

Attackers can gain access to sensitive information through API errors.

### Recommendation

Never expose raw API errors to users. Instead, return a generic error message

### POC



```

src > contexts > CustomerContext.tsx > CustomerProvider > createCustomer > useCallback() callback
31 export function CustomerProvider({ children }: { children: ReactNode }) {
32 const getCustomer = useCallback(async (id: string) => {
33 const response = await fetch(` /api/customers/${id}`);
34 if (!response.ok) {
35 throw new Error('Failed to fetch customer');
36 }
37 const data = await response.json();
38 return data.customer;
39 }, [customers]);
40
41 const createCustomer = useCallback(async (data: { name: string; email: string; address: string }) => {
42 const response = await fetch(` /api/customers`, {
43 method: "POST",
44 headers: { "Content-Type": "application/json" },
45 body: JSON.stringify(data),
46 });
47
48 if (!response.ok) {
49 const error = await response.text();
50 throw new Error(error || 'Failed to create customer');
51 }
52
53 const newCustomer = await response.json();
54 setCustomers(prev => prev ? [...prev, newCustomer] : [newCustomer]);
55 return newCustomer;
56 }, []);
57
58 const updateCustomer = useCallback(async (id: string, data: { name: string; email: string; address: string }) => {
59 const response = await fetch(` /api/customers/${id}`, {
60 method: "PUT",
61 headers: { "Content-Type": "application/json" },
62 body: JSON.stringify(data),
63 });
64
65 if (!response.ok) {
66 const error = await response.text();
67 throw new Error(error || 'Failed to update customer');
68 }
69
70 const updatedCustomer = await response.json();
71 setCustomers(prev =>
72 prev ? prev.map(c => c.id === id ? updatedCustomer : c) : [updatedCustomer]
73);
74
75 return updatedCustomer;
76 }, []);
77
78 };
79
80 return (
81 <Provider value={CustomerProvider}>{children}</Provider>
82);
83}
84
85
```

Ln 72, Col 1 Spaces: 2 UTF-8 LF {} TypeScript JSX

## Outdated Server Components

Resolved

### Description

The application is using [github.com/redis/go-redis/v9](https://github.com/redis/go-redis/v9) v9.7.0 which is outdated and vulnerable to CVE-2025-2992.

### Vulnerable File

go.mod

### Impact

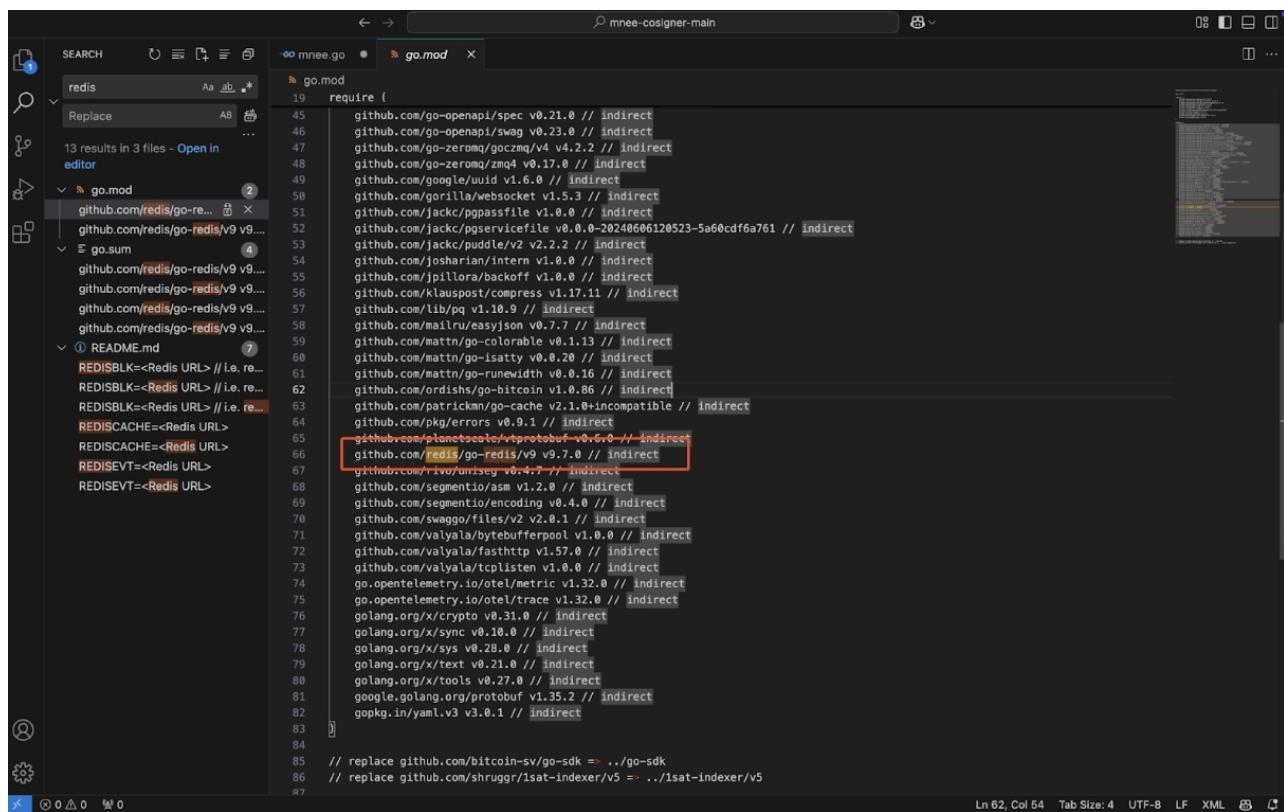
Update the dependency to the latest stable secure version.

### Recommendation

The dependency has known exploit available. Attackers can exploit the known vulnerability.

### POC

The dependency has known exploit available. Attackers can exploit the known vulnerability.



A screenshot of a code editor interface showing the go.mod file for the mnee-cosigner-main project. The file contains the following content:

```

require (
 github.com/go-openapi/spec v0.21.0 // indirect
 github.com/go-openapi/swag v0.23.0 // indirect
 github.com/go-zeromq/goczm/v4 v4.2.2 // indirect
 github.com/go-zeromq/zmq4 v0.17.0 // indirect
 github.com/google/uuid v1.6.0 // indirect
 github.com/jackc/pgservicefile v0.0.0-20240606120523-5a60cdf6a761 // indirect
 github.com/jackc/pgservicefile/v2 v2.2.2 // indirect
 github.com/jackc/puddle/v2 v2.2.2 // indirect
 github.com/josharian/interv v1.0.0 // indirect
 github.com/jpillora/backoff v1.0.0 // indirect
 github.com/klauspost/compress v1.17.11 // indirect
 github.com/lib/pq v1.10.9 // indirect
 github.com/mailru/easyjson v0.7.7 // indirect
 github.com/matttn/go-colorable v0.1.13 // indirect
 github.com/matttn/go-runewidth v0.0.16 // indirect
 github.com/ordisshs/go-bitcoin v1.0.86 // indirect
 github.com/patrickmn/go-cache v2.1.0+incompatible // indirect
 github.com/pkg/errors v0.9.1 // indirect
 github.com/plantastic/viprotobuf v0.6.0 // indirect
 github.com/redis/go-redis/v9 v9.7.0 // indirect
 github.com/rivo/uniseg v0.4.7 // indirect
 github.com/segmentio/asn v1.2.0 // indirect
 github.com/segmentio/encoding v0.4.0 // indirect
 github.com/swaggo/files/v2 v2.0.1 // indirect
 github.com/valyala/bytbufpool v1.0.0 // indirect
 github.com/valyala/fasthttp v1.57.0 // indirect
 github.com/valyala/tcplisten v1.0.0 // indirect
 go.opentelemetry.io/otel/metric v1.32.0 // indirect
 go.opentelemetry.io/otel/trace v1.32.0 // indirect
 golang.org/x/crypto v0.31.0 // indirect
 golang.org/x/sync v0.10.0 // indirect
 golang.org/x/sys v0.28.0 // indirect
 golang.org/x/text v0.21.0 // indirect
 golang.org/x/tools v0.27.0 // indirect
 google.golang.org/protobuf v1.35.2 // indirect
 gopkg.in/yaml.v3 v3.0.1 // indirect
)

// replace github.com/bitcoin-sv/go-sdk => ../go-sdk
// replace github.com/shruggr/isat-indexer/v5 => ../isat-indexer/v5

```

The line `github.com/redis/go-redis/v9 v9.7.0 // indirect` is highlighted with a red rectangle, indicating it is the target of the exploit.

# Closing Summary

In this report, we have considered the security of the MNEE. We performed our audit according to the procedure described above.

Some issues of High, medium, low, and Informational severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

# Disclaimer

QuillAudits Dapp audit is not a security warranty, investment advice, or an endorsement of the MNEE. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multi-step process. One audit cannot be considered enough.

We recommend that the MNEE Team put in place a bug bounty program to encourage further analysis of the source code by other third parties.

# About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers.

With seven years of expertise, we've secured over 1400 projects globally, averting over \$3 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.



|                                 |                                     |
|---------------------------------|-------------------------------------|
| <b>7+</b><br>Years of Expertise | <b>1M+</b><br>Lines of Code Audited |
| <b>50+</b><br>Chains Supported  | <b>1400+</b><br>Projects Secured    |

Follow Our Journey



# AUDIT REPORT

---

October 2025

For



 QuillAudits

Canada, India, Singapore, UAE, UK

[www.quillaudits.com](http://www.quillaudits.com)

[audits@quillaudits.com](mailto:audits@quillaudits.com)