Cyberscope

A TAC Security Company

Audit Report

Celestia

November 2025

Repository https://github.com/HEIN-TECH/celestia.finance

Commit 07bef3b94efbf7742a7428fb062787cb3c9ee20b

Audited by © cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

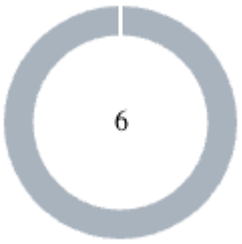
Review

Repository	https://github.com/HEIN-TECH/celestia.finance
Commit	07bef3b94efbf7742a7428fb062787cb3c9ee20b

Audit Updates

Initial Audit	03 Nov 2025
Corrected Phase 2	14 Nov 2025

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	6

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	6	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	IRLL	In-Memory Rate Limiting Limitations	Unresolved
	ICC	Insecure CORS Configuration	Unresolved
	MNTC	Magic Number Time Conversion	Unresolved
	MIV	Missing Input Validation	Unresolved
	PIAE	Potential IP Address Exposure	Unresolved
	SESE	Sensitive Error Stack Exposure	Unresolved

IRLL - In-Memory Rate Limiting Limitations

Criticality	Minor / Informative
Location	src/frontend/src/lib/rate-limit.ts
Status	Unresolved

Description

The rate limiting system relies on an in-memory Map to track requests and account lockouts. This approach works only on a single long-running server instance. In serverless environments such as Vercel, AWS Lambda, or Cloudflare Workers, each function invocation runs in an isolated, stateless runtime. Memory does not persist between requests, meaning the rate limiter resets on every invocation. As a result, the application has no effective protection against brute-force attacks, login abuse, or high-volume API calls when deployed serverlessly. This creates a significant security gap, as attackers can bypass rate limits simply by triggering new serverless instances.

```
class RateLimiter {
  private store: Map<string, RateLimitEntry>;
  private accountLocks: Map<string, AccountLockEntry>; // 账户锁定存储
  private cleanupInterval: NodeJS.Timeout | null;

  constructor() {
    this.store = new Map();
    this.accountLocks = new Map();
    this.cleanupInterval = null;
    this.startCleanup();
  }
  ...
}
```

Recommendation

The team is advised to replace the in-memory rate limiting implementation with a persistent, distributed rate limiter that works across serverless instances.

The team could use one of the following:

- Upstash Redis + @upstash/ratelimit (recommended for Vercel)
- Redis with a sliding window or token bucket algorithm
- Cloudflare KV / Durable Objects

By using a shared data store, rate limits persist across all serverless invocations and deployments, ensuring robust defense against brute-force and high-frequency attacks.

ICC - Insecure CORS Configuration

Criticality	Minor / Informative
Location	src/backend/main.py#L41
Status	Unresolved

Description

The CORS middleware is configured to allow all origins (*) via the allow_origins parameter. This configuration effectively permits any domain to send cross-origin requests to the API, which can expose sensitive data to unauthorized third-party websites. When combined with `allow_credentials=True`, it creates a severe security risk — attackers could execute Cross-Site Request Forgery (CSRF) or data exfiltration attacks from untrusted origins, as cookies and authentication tokens might be automatically sent with the requests.

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=ALLOWED_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
    expose_headers=["*"],  # 暴露所有响应头
    max_age=3600,  # 预检请求缓存时间
)
```

Recommendation

Restrict CORS access to trusted domains only. Avoid using wildcard (*) origins, especially when allow_credentials=True is enabled.

- Replace `allow_origins=["*"]` with a list of known and trusted frontend domains
- Keep allow_credentials=True only if absolutely necessary for authenticated cross-origin requests.
- If the API is public and must support multiple origins, implement dynamic origin validation (check the Origin header against an approved whitelist at runtime).

Properly restricting CORS origins ensures that only authorized web applications can interact with the API, reducing the risk of cross-site attacks.

MNTC - Magic Number Time Conversion

Criticality	Minor / Informative
Location	src/frontend/src/lib/utils.ts#L8
Status	Unresolved

Description

The function uses the hardcoded value `1e12` to determine whether a timestamp is in seconds rather than milliseconds. This magic number lacks context, making the code less readable and harder to maintain. Without a clear explanation, future developers may not understand its purpose or origin.

```
export const getElapsedTimeLabel = (timestamp: number) => {
  if (timestamp < 1e12) {
    timestamp *= 1000;
  }

  const now = Date.now();
  const diffMs = now - timestamp;
  const diffMin = Math.floor(diffMs / (1000 * 60));
  const diffH = Math.floor(diffMin / 60);
  const diffD = Math.floor(diffH / 24);
  const diffW = Math.floor(diffD / 7);

  if (diffW >= 1) {
    return `${diffW}w`;
  } else if (diffD >= 1) {
    return `${diffD}d`;
  } else if (diffH >= 1) {
    return `${diffH}h`;
  } else {
    return `${diffMin}min`;
  }
};
```

Recommendation

Define named constants such as `SECONDS_TO_MILLISECONDS_THRESHOLD` and include comments that clarify the purpose and logic behind the conversion. This improves code readability and helps future developers understand the rationale behind the threshold values.

MIV - Missing Input Validation

Criticality	Minor / Informative
Location	src/frontend/src/app/api/invite/verify/route.ts#L102 src/backend/email_router.py#L10,91,109,127
Status	Unresolved

Description

The request body is parsed directly without any form of validation, which can lead to unexpected behavior or security vulnerabilities if the input is malformed or malicious.

```
const body = await request.json();
const { code, userEmail } = body;
```

```
customer_task = asyncio.create_task(
    email_service.send_customer_email(email_data)
)
admin_task = asyncio.create_task(
    email_service.send_admin_notification(email_data)
)
customer_result, admin_result = await asyncio.gather(
    customer_task, admin_task, return_exceptions=True
)
```

Recommendation

By implementing request validation, all input parameters can be rigorously checked for type correctness and required structure. This ensures that only well-formed data is processed by the application, reducing the risk of runtime errors and unexpected behavior.

Additionally, sanitizing user input helps prevent injection attacks and other forms of malicious input, enhancing the overall security and reliability of the system.

PIAE - Potential IP Address Exposure

Criticality	Minor / Informative
Location	src/frontend/next.config.ts#L15
Status	Unresolved

Description

The `allowedDevOrigins` array includes production IP addresses, which may inadvertently expose development-only endpoints in a live environment. This configuration increases the risk of unauthorized access to internal tools or untested features, potentially compromising application security.

```
const nextConfig: NextConfig = {
  distDir: 'hope-fund-frontend',
  reactStrictMode: true,

  transpilePackages: [
    '@ant-design',
    'antd',
    'rc-util',
    'rc-pagination',
    'rc-picker',
  ],

  allowedDevOrigins: [
    'https://45.77.170.234:3000',
    'https://45.77.170.234',
    'http://localhost:3000',
    'http://127.0.0.1:3000',
    'http://192.168.50.14:3000',
  ],
  ...
};
```

Recommendation

The team is advised to remove production IP addresses from development origins. This helps prevent unintended access to production resources during development.

Additionally, using environment-based configuration ensures that settings are appropriately tailored for development, staging, or production environments, reducing the risk of misconfiguration.

Finally, dev-only features should be properly gated to prevent exposure in production, enhancing security and maintaining a clean user experience.

SESE - Sensitive Error Stack Exposure

Criticality	Minor / Informative
Location	src/frontend/src/app/api/basic-info/route.ts#L204
Status	Unresolved

Description

The API route returns the full `error.stack` trace in its JSON response when an exception occurs. This exposes internal implementation details — such as file paths, function names, and potentially sensitive data — to clients. Attackers can use this information to map the server structure, identify frameworks or libraries in use, and craft targeted exploits. Exposing stack traces is a common security misconfiguration that violates secure error-handling best practices.

```
console.error('❌ API Error:', error);
return NextResponse.json(
  {
    error: 'Failed to load vault data',
    details: error.message,
    stack: error.stack
  },
  { status: 500 }
);
```

Recommendation

The team is advised to remove the `stack` and, ideally, the `details` fields from production error responses. Full error information should be logged securely on the server side using a reliable logging system such as Winston, Pino, or Sentry. Only a generic, non-sensitive error message should be returned to the client to prevent exposure of internal implementation details.

Summary

Celestia implements a backend and frontend mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a TAC blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

A **TAC Security** Company

The Cyberscope team

cyberscope.io