shieldify

Colb Finance

Web2 Audit

SECURITY REVIEW

Date: 17 December 2025

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and have secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Vyper, Rust, Cairo, Move and Go.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Colb Finance - Web2 Audit

Colb is the first native non-custodial tokenization solution that enables peerless access to Swiss-grade wealth management strategies, pre-IPO opportunities, and premium investment funds. It offers a bankruptcy remote Trust structure and native ownership of real-world assets, all on-chain. Colb reduces the entrance threshold to such investments by removing constraints such as the minimum investment amount to get exposure to them. The protocol is designed with security at its core, boasting compliance with Swiss regulations and DeFi composability. Colb envisions a future rooted in transparency where every individual has equitable access to premium RWA investments.

4. Risk Classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** - results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** - results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** - losses will be limited but bearable - and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** - almost certain to happen and highly lucrative for execution by malicious actors

- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted 7 days with a total of 56 hours dedicated to the audit by the Shieldify team.

Overall, the code is well-written. The audit report contributed by identifying one High, one Medium and eight Low severity issues, mainly related to unauthenticated user access, missing automatic wallet account change and missing input validation in different areas.

The Colb Finance team has done a great job with their test suite and provided support and responses to all of the questions that the Shieldify researchers had.

5.1 Protocol Summary

Project Name	**Colb Finance - Web2**
Repository	ColbUI
Type of Project	User Interface, Administrator Back Office
Security Review Timeline	7 days
Review Commit Hash	979c7f20fa667cdba9b402d4b8ad1f26472cfca3
Fixes Review Commit Hash	f5c3fbb821f998ec5d66a09030c8c6c5df43ce29

5.2 Scope

The files from the following folders were in the scope of the security review:

Folders
main/apps/backoffice
main/apps/frontend

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **High** issues: **1**
- **Medium** issues: **1**
- **Low** issues: **8**
- **Info** issues: **2**

ID	Title	Severity	Status
[H-01]	Unauthenticated Access Token Issuance in `SumSub` API	High	Fixed
[M-01]	Missing Automatic Wallet Account Change Handling in Dapp	Medium	Fixed
[L-01]	Lack of Input Validation Across Multiple Authenticated JSON Endpoints	Low	Fixed
[L-02]	Lack of Server-Side Validation Allows Tampering with Pool Exchange Rates in the Dapp	Low	Fixed
[L-03]	Lack of Input Validation Allows Negative Maturity Term in Pools	Low	Fixed
[L-04]	Lack of Input Validation Allows Tampering of Cash-out Request Dates (Polygon Amoy Network)	Low	Fixed
[L-05]	Missing Input Length Restriction in `Reject Request` Comment Field of Buy `USC` Manager	Low	Fixed
[L-06]	Missing Input Length Restriction in `Reject Request` Comment Field – Polygon Amoy Network (Cash `USC` Manager)	Low	Fixed
[L-07]	Prisma Schema and Internal Metadata Exposure in Client JavaScript Bundle	Low	Acknowledged
[L-08]	Method Handling Strictness (Applies to ALL API files)	Low	Fixed
[I-01]	Unhandled Blockchain Transaction Errors Expose Low-Level Debug Data in UI	Info	Fixed
[I-02]	Undefined Error Handling in JWT Verification	Info	Fixed

7. Findings

[H-01] Unauthenticated Access Token Issuance in `SumSub` API

Severity

High Risk

Description

The `/app/api/sumsub-access-token` endpoint issues valid Sumsub access tokens without requiring any authentication or authorization. The endpoint accepts two query parameters:

- `userId` – the blockchain wallet address or unique identifier of the target account
- `levelName` – the KYC verification level (e.g., `BASIC`, `CORPORATE`)

An attacker can manipulate both parameters to: Impersonate any user by providing their `userId`. Potentially manipulate roles by changing `levelName` from BASIC to CORPORATE or other higher tiers.

This grants the attacker a valid JWT for the Sumsub API/SDK, which might be used in the flow for forged requests. The following PoC confirms the issue:

1. Request `BASIC` Level Access Token

```
curl -k "https://app.testnet.colb.io/app/api/sumsub-access-token?
   userId=0x6682490d38eb880bfbb5ef132be3b56fabc0832f&levelName=BASIC"
```

2. Returns a valid Sumsub token for `BASIC` verification.
3. Go to `CORPORATE` Level

```
curl -k "https://app.testnet.colb.io/app/api/sumsub-access-token?
   userId=0x6682490d38eb880bfbb5ef132be3b56fabc0831f&levelName=
   CORPORATE"
```

Response:

```
{
  "token": "_act-sbx-jwt-eyJhbGciOiJub25lIn0.eyJq
  dGkiOiJfYWN0LXNieC0zMjhjZDFmZC01MWIxLTQ5MDA
  tOGFjYy00YzZiYTJhYTdjMmItdjIiLCJ1cmwiOiJodH
  RwczovL2FwaS5zdW1zdWIuY29tIn0.-v2"
}
```

Impact

Integrity compromise: Ability to upload forged tokens and potentially pass fraudulent verification. Level Name manipulation: `BASIC` -level users can potentially obtain `CORPORATE` -level access tokens without approval.

Location of Affected Code

Link: /app/api/sumsub-access-token

Justification by OWASP:

OWASP API Security Top 10 – API2:2023 – Broken Authentication The endpoint grants valid Sumsub access tokens without requiring authentication. This violates the principle that sensitive API actions, such as initiating or continuing identity verification flows, must be protected by identity checks.

OWASP API Security Top 10 – API5:2023 – Broken Function-Level Authorization There are no controls to verify that the user making the request has the right to request a Sumsub token for the specified `userId`, nor that the requested `levelName` is appropriate. This allows escalation of privileges and impersonation.

OWASP API Security Top 10 – API8:2023 – Security Misconfiguration The fact that sensitive GET endpoints like this are exposed without access control reflects a misconfigured API environment. The system should follow the principle of least privilege and not allow open issuance of authentication tokens via GET requests.

This finding highlights an important flaw in how GET requests are handled across the application. The `/sumsub-access-token` endpoint allows unauthenticated users to obtain valid tokens by simply guessing or supplying another user's `userId` and `level`. This is part of a broader architectural issue where sensitive API operations (e.g., whitelist checks, transaction data, Sumsub integrations) are exposed via unprotected GET routes, effectively nullifying identity enforcement in the platform and enabling impersonation or potential privilege escalation attacks.

Additionally, this flaw can be chained to tamper with the Whitelist process of a wallet, triggering a forged Profile verification successful in the UI.

Recommendation

Require authentication before issuing any KYC access token. Consider implementing object-level authorization checks to ensure that:

- The `userId` belongs to the authenticated user.
- The requested `levelName` matches the user's permitted verification level.
- Reject any requests where `levelName` is manually supplied; derive it server-side from the account configuration.

Team Response

Fixed.

[M-01] Missing Automatic Wallet Account Change Handling in Dapp

Severity

Medium Risk

Description

The back-office Dapp (and the `app.testnet`) connected to MetaMask does not automatically handle wallet account changes. When the user connects with Account 1 (`0x6682490d38eB880bFbb5Ef132Be3b56fAbC0832F`) and then switches to Account 2 (`0x7e49d3eb1Ba82BB65d0FCe029E712612656710bF`) directly in MetaMask, the Dapp continues to display the original account without triggering a reconnection prompt or UI update. This results in a mismatch between the wallet currently active in the browser extension and the wallet recognized by the Dapp's frontend/backend. When performing a transaction, MetaMask detects this mismatch and blocks the transaction, but the Dapp itself does not detect or prevent it.

1. Connect the Dapp to MetaMask using Account 1.
2. In MetaMask, switch to Account 2 without reconnecting to the Dapp.
3. Observe that the UI still displays Account 1.
4. Attempt an operation (e.g., `Add Liquidity`).

MetaMask prompts: `Your wallets are mismatched. Switch to continue.`

6. The Dapp never updates or prompts for reconnection automatically.

Justification by OWASP:

OWASP ASVS 4.0 – V4: Access Control "V4.1.3: Verify that access controls fail securely, including when an exception occurs." The Dapp fails to enforce secure handling when the connected account changes unexpectedly. Although MetaMask blocks the transaction, the Dapp should detect the mismatch and fail gracefully by requiring user reconnection.

OWASP API Security Top 10 – API3:2023 – Broken Object Property Level Authorization While the mismatch is client-side, continuing to display the old account may lead the user to believe their new wallet is authorized when it's not. This reflects poor frontend authorization awareness tied to sensitive identity-related operations.

OWASP Top 10 – A5:2021 – Security Misconfiguration The failure to listen for account change events and re-establish secure sessions (via `walletListener` or `onAccountsChanged()`) reflects a misconfiguration in Web3 session management that could lead to UX or logic inconsistencies.

Impact

User Confusion – Users may unintentionally attempt actions assuming the Dapp is linked to the new wallet, causing failed transactions. Business Process Disruption – Flows that depend on a specific whitelisted address (KYC, liquidity pool operations) may be bypassed in the UI until MetaMask rejects the transaction.

Location of Affected Code

Link: backoffice.testnet.colb.io

Link: backoffice.testnet.colb.io

Recommendation

Implement event listeners for `accountsChanged` in the Web3 provider (`ethereum.on('accountsChanged', handler)`). On account change:

- Prompt the user to reconnect.
- Automatically refresh session data to match the new account.
- Invalidate any pending transactions tied to the old account.

Team Response

Fixed.

[L-01] Lack of Input Validation Across Multiple Authenticated JSON Endpoints

Severity

Low Risk

Description

Multiple authenticated API endpoints in both the back office and frontend accept unvalidated JSON input, allowing special characters, malformed values, and type inconsistencies to be submitted.

Examples include:

- POST `/backoffice/api/pools-exchange-rates`
- PATCH `/backoffice/api/cash-scb-transaction`

In both cases:

Parameters such as `rate`, `createdAt`, `poolType`, `id`, and `comment` accept arbitrary strings, special characters, or malformed date formats without being rejected at the application layer.

The backend eventually returns an error (400 Bad Request),

```
Invalid data provided: {"poolType":"Balance","rate":1,"createdAt":"
    2025'-08-11T10:40:03.570Z"}
```

But the frontend UI becomes stuck in a loading state, requiring manual refresh. This demonstrates a lack of server-side schema validation and consistent error handling.

It is important to note that while the application uses `Prisma.sql` for database interactions, reducing direct SQL injection risk, this lack of validation still poses a platform-wide integrity and availability risk if these values are processed by other components (e.g., logging, rendering, third-party integrations) in the future. The following PoC illustrates the issue:

Request:

```
POST /backoffice/api/pools-exchange-rates HTTP/2
Host: backoffice.testnet.colb.io
Content-Type: application/json
Token: <valid-JWT>

{"poolType":"Balance","rate":1'","createdAt":"2025'-08-11T10:40:03.570Z"}
```

Response:

```
HTTP/2 400 Bad Request
Invalid data provided: {"poolType":"Balance","rate":1,"createdAt":"
    2025'-08-11T10:40:03.570Z"}
```

Frontend behavior: The Interface remains loading indefinitely until the page refreshes.

Justification by OWASP:

- OWASP API Security Top 10 – API8:2023 – Injection – Even without immediate SQL injection, unvalidated input expands the injection attack surface for future features.
- OWASP API Security Top 10 – API4:2023 – Unrestricted Resource Consumption – Malformed or oversized input can cause excessive processing, leading to availability issues.
- OWASP ASVS 5.3.2 – "Verify that all parameters are validated using a positive security model."

The high severity comes from the fact that this is a systemic input validation flaw affecting multiple backend endpoints (for additional context and support of this severity see findings Lack of Server-Side Validation Allows Tampering with Pool Exchange Rates in the Dapp and Lack of Input Validation Allows Tampering of Cash-out Request Dates for example), impacting data integrity and application availability. The issue can be abused platform-wide to cause UI disruption, probe for unhandled payloads, and potentially escalate if processed by downstream components in the future.

Impact

- Integrity Risk – Malformed data could be inserted if future code changes allow partial processing before rejection.
- Availability Risk – Current behavior disrupts workflows by freezing the UI until manual refresh.
- Systemic Attack Surface – Multiple endpoints share the same flaw, meaning attackers can probe for unhandled payloads platform-wide.
- Future Exploit Potential – If values are later reused without escaping, this could escalate to XSS, SSRF, or even RCE in downstream systems.

Location of Affected Code

Routes:

- POST `/backoffice/api/pools-exchange-rates` (`rate` , `poolType` , `createdAt`)
- PATCH `/backoffice/api/cash-scb-transaction` (`id` , `status` , `completedAt` , `comment`)

It's likely present in other JSON-based POST/PATCH endpoints without schema enforcement.

Recommendation

- Implement strict server-side input validation for all API payloads using a schema validation library (e.g., Joi, Zod, or built-in Prisma Zod validation).
- Validate type, format, and range for numeric, date, and string fields.
- Reject malformed requests early with meaningful error messages.
- Ensure frontend handles backend validation errors gracefully without freezing.

Team Response

Fixed.

[L-02] Lack of Server-Side Validation Allows Tampering with Pool Exchange Rates in the Dapp

Severity

Low Risk

Description

The backoffice endpoint `/backoffice/api/pools-exchange-rates` accepts POST requests that directly update exchange rate records for investment pools. The server does not validate the rate or `createdAt` parameters, allowing an attacker to submit arbitrary values, including negative rates and impossible dates far in the future.

In the following PoC, the following parameters were modified:

- rate to `-100000`
- createdAt to `5025-08-06T11:00:08.889Z`

These changes were reflected in the Managed Pools interface under Earn Products, confirming that the tampering is stored and displayed without sanitization.

Vulnerable POST request example:

```
POST /backoffice/api/pools-exchange-rates HTTP/2
Host: backoffice.testnet.colb.io
Content-Type: application/json
Token: <valid_jwt>

{
  "poolType": "Balance",
  "rate": -100000,
  "createdAt": "5025-08-06T11:00:08.889Z"
}
```

The tampered parameters were processed in the UI. No server-side rejection or normalization occurred:

Justification by OWASP:

OWASP API Security Top 10 – API6:2023 – Unrestricted Access to Sensitive Business Flows The endpoint allows direct manipulation of financial records without adequate safeguards. Sensitive operations like updating investment exchange rates must be protected by strict business logic validations, input constraints, and user privilege checks.

OWASP API Security Top 10 – API8:2023 – Security Misconfiguration The lack of server-side validation for critical fields such as `rate` and `createdAt` represents a misconfiguration. Accepting unchecked inputs into core business logic is a failure of secure-by-default design principles.

OWASP API Security Top 10 – API9:2023 – Improper Inventory Management There is no evidence of endpoint classification or access restriction based on sensitivity. A high-impact operation is exposed without logging, throttling, or validation layers, suggesting a lack of asset inventory and protection based on data sensitivity.

Impact

Data Integrity Compromise: Tampering with the financial rate record. Reputation Damage: Public dashboards or investor reports could display misleading figures.

Location of Affected Code

File: backoffice/api/pools-exchange-rates

Link: backoffice.testnet.colb.io

Link: backoffice.testnet.colb.io

Recommendation

Server-Side Validation:

- Reject negative rates or rates exceeding realistic bounds.
- Enforce valid date ranges (`createdAt` should be within a reasonable timeframe).

Team Response

Fixed.

[L-03] Lack of Input Validation Allows Negative Maturity Term in Pools

Severity

Low Risk

Description

The `/backoffice/api/pool-maturity-term` endpoint accepts PATCH requests to set the maturity term of a pool. The server does not validate the `term` parameter, allowing submission of negative values or unrealistic durations.

We modified the maturity term by sending: `{"term": -150}`

This change was accepted and applied without error, potentially altering timelines in a way that contradicts business logic.

Vulnerable Request Example

```
PATCH /backoffice/api/pool-maturity-term?poolType=0 HTTP/2
Host: backoffice.testnet.colb.io
Content-Type: application/json
Token: <valid_jwt>

{"term": -150}
```

Justification by OWASP:

OWASP API Security Top 10 – API6:2023 – Unrestricted Access to Sensitive Business Flows The endpoint allows unrestricted manipulation of a key financial parameter (`term`) without validation, directly impacting business logic. Attackers could alter investment pool behavior (e.g., immediate maturity or indefinite delay), bypassing expected application workflows.

OWASP Web Security Testing Guide – 6.6.1 Business Logic Testing This case falls squarely within business logic abuse, altering pool terms to unrealistic values can result in incorrect fund release, reporting, or calculations, undermining platform integrity.

Location of Affected Code

File: pool-maturity-term.ts

Link: app.testnet.colb.io/app/balance

Link: backoffice.testnet.colb.io

Impact

Business Logic Violation: Pools can end in the past or have nonsensical maturity periods.

Recommendation

Server-Side Validation:

- Enforce the `term` to be a positive integer within an acceptable range.
- Reject zero, negative, or excessively large values.

Team Response

Fixed.

[L-04] Lack of Input Validation Allows Tampering of Cash-out Request Dates (Polygon Amoy Network)

Severity

Low Risk

Description

On the Polygon Amoy network, the cash-scb-transaction endpoint processes approval/rejection actions for cashout requests. When rejecting a request, the backend accepts and stores the `completedAt` timestamp supplied by the client without any validation. It is possible to submit a `completedAt` value far in the future (8025-08-06T11:54:17.509Z), and the system:

- Accepts the value without error.
- Persists it in the database.
- Displays it directly in the Cash $USC Requests interface.

Vulnerable Request Example

```
PATCH /backoffice/api/cash-scb-transaction HTTP/2
Host: backoffice.testnet.colb.io
Content-Type: application/json
Token: <valid_jwt>

{
  "id": 122,
  "status": "Rejected",
  "completedAt": "8025-08-06T11:54:17.509Z",
  "comment": ";"
}
```

Justification by OWASP:

OWASP API Security Top 10 – API6:2023 – Unrestricted Access to Sensitive Business Flows The endpoint allows manipulation of business-critical metadata (`completedAt`) without server-side enforcement of business logic boundaries. This reflects a failure to control key workflows that can be abused to falsify system state or disrupt financial operations.

OWASP API Security Top 10 – API8:2023 – Security Misconfiguration Accepting arbitrary timestamps directly from the client, without validating for realism, boundaries, or correctness, represents a security misconfiguration. The server trusts input that should be tightly controlled.

OWASP API Security Top 10 – API3:2023 – Broken Object Property Level Authorization By allowing direct manipulation of `completedAt` , users gain undue influence over internal transaction attributes that should be restricted or automatically set by backend logic.

Impact

Data Integrity Compromise: Malicious users can insert nonsensical or fraudulent timestamps into critical transaction records on Polygon Amoy. Audit Trail Corruption: Misleading transaction dates hinder investigations, audits, and compliance verification.

Location of Affected Code

File: cash-scb-transaction.ts

Link: backoffice.testnet.colb.io

Link: backoffice.testnet.colb.io

Recommendation

Server-Side Validation:

- Ensure `completedAt` dates are within a reasonable range (e.g., ±1 year from the current date).
- Reject negative or far-future dates.

Team Response

Fixed.

[L-05] Missing Input Length Restriction in `Reject Request` Comment Field of Buy $USC Manager

Severity

Low Risk

Description

In the Buy $USC Manager module, the `comment` field within the reject request functionality does not enforce any length limit. A user can submit arbitrarily long text strings, which are then fully rendered in the tooltip in the UI. This leads to:

- Tooltip that disrupts layout and usability.
- Potential excessive database storage if the backend does not validate or truncate the input.
- Performance degradation in both frontend rendering and backend processing when handling large payloads.
- If such input is stored and later displayed without proper encoding, it could also become an injection vector (e.g., HTML/JavaScript execution).

The following PoC illustrates the issue:

1. In the Buy $USC Manager interface, trigger the reject request workflow.
2. In the `comment` field, paste an excessively large text block (e.g., >10,000 characters).
3. Submit the request.
4. Hover over the relevant tooltip in the UI and observe that the full text displays, breaking layout and scrolling behavior.

Justification by OWASP:

OWASP Top 10 – A01:2021 – Broken Access Control (UI manifestation) Although not a classic access control issue, failing to limit and sanitize user input can result in unintended behavior or exposure of system internals via improperly handled front-end components like tooltips.

Impact

UI Disruption, oversized tooltips overlap and break surrounding UI components. Performance and Storage Issues: Large inputs may consume excessive database space, increase transfer sizes, and slow down rendering.

Location of Affected Code

Link: app.testnet.colb.io/app/balance

Link: backoffice.testnet.colb.io

Recommendation

- Enforce a maximum input length for the `comment` field (e.g., 255–500 characters) on both frontend and backend.
- Truncate or summarize the tooltip display to prevent UI disruption.

Team Response

Fixed.

[L-06] Missing Input Length Restriction in `Reject Request` Comment Field – Polygon Amoy network (Cash $USC Manager)

Severity

Low Risk

Description

In the Polygon Amoy network, Cash $USC Manager modules allow rejected request comments with no maximum length enforcement. Arbitrarily long text is accepted, stored, and rendered without truncation. The oversized strings fully display in tooltips or table cells, disrupting the UI layout and potentially causing performance degradation.

The following PoC illustrates the issue:

1. Go to the Cash $USC Manager module in the Polygon Amoy network..
2. Open a reject request form and insert a long text string (e.g., several thousand characters).
3. Submit the form and reload the page.
4. Observe that the comment renders in full in the table and tooltip, breaking layout (see provided screenshots).

Justification by OWASP:

OWASP Top 10 – A01:2021 – Broken Access Control (UI manifestation) Although not a classic access control issue, failing to limit and sanitize user input can result in unintended behavior or exposure of system internals via improperly handled front-end components like tooltips.

Impact

UI Disruption, oversized tooltips overlap and break surrounding UI components. Performance and Storage Issues: Large inputs may consume excessive database space, increase transfer sizes, and slow down rendering.

Location of Affected Code

Link: app.testnet.colb.io/app/balance

Link: backoffice.testnet.colb.io/

Recommendation

- Enforce a maximum input length for the comment field (e.g., 255–500 characters) on both frontend and backend.
- Truncate or summarize the tooltip display to prevent UI disruption.

Team Response

Fixed.

[L-07] Prisma Schema and Internal Metadata Exposure in Client JavaScript Bundle

Severity

Low Risk

Description

Two publicly accessible JavaScript bundles in the production environment expose internal Prisma ORM schema definitions, entity names, enum values, and Prisma client metadata:

- https://backoffice.testnet.colb.io/backoffice/_next/static/chunks/pages/_app-a34654c57e274261.js
- https://app.testnet.colb.io/app/_next/static/chunks/pages/_app-32ae95f8e10545f7.js

These files contain:

- Full Prisma ScalarFieldEnum definitions revealing database table and column names (`userWallet` , `accountNumber` , `smartContractAddress` , `status` , `penaltyFee` , etc.).
- Prisma client version (6.3.0) and engine hash (`acc0b9dd43eb689cbd20c9470515d719db10d0b0`) enabling precise technology fingerprinting.
- Internal data models for various entities (e.g., `BuyScbTransaction` , `Swap` , `CancelRequest` , `InvestmentAccess`).
- Error handling messages that disclose expected database behaviors and ORM operations.
- This information is not required for frontend operation and unnecessarily increases the attack surface.

```
client:"6.3.0", engine:"acc0b9dd43eb689cbd20c9470515d719db10d0b0"
```

Justification by OWASP:

OWASP API Top 10 – API9:2023 Improper Assets Management Sensitive backend assets (ORM schema, internal enums) are accessible in production without restriction. OWASP Top 10 – A05:2021 Security Misconfiguration Insecure build configuration allowed server-side code and schema metadata to leak to the public frontend.

Impact

- An attacker could:
- Enumerate database schema and relationships without authentication.
- Craft more precise SQL/ORM injection payloads using known entity and field names.
- Identify Prisma version-specific vulnerabilities for targeted exploitation.

Location of Affected Code

File: _app-32ae95f8e10545f7.js

File: _app-a34654c57e274261.js

Recommendation

Disable source map exposure in production.

Team Response

Acknowledged.

[L-08] Method Handling Strictness (Applies to ALL API files)

Severity

Low Risk

Description

The default case in every switch (`req.method`) uses:

```
res.status(StatusCodes.NOT_IMPLEMENTED).send(ReasonPhrases.
    NOT_IMPLEMENTED);
```

Instead of:

- Returning 405 Method Not Allowed for unsupported but understood HTTP methods.
- Including an Allow header listing supported methods.

Justification by OWASP:

OWASP API Security Top 10 – API8:2023 – Security Misconfiguration Returning incorrect HTTP status codes, such as 501 Not Implemented for unsupported methods, is a classic example of misconfiguration. It misleads clients about the API's capabilities and violates protocol standards, which can cause unintended behavior in automated tooling or API consumers.

OWASP ASVS 4.0 – V9: Communication "V9.1.1: Verify that HTTP response headers and status codes are used appropriately and according to standards." The misuse of HTTP status codes fails this control, as 405 Method Not Allowed is the correct response when the method is known but not supported by the endpoint, and the Allow header should be used to indicate supported methods.

OWASP Top 10 – A5:2021 – Security Misconfiguration This is a case of poor adherence to HTTP semantics, which can degrade the interoperability, debuggability, and security posture of the API.

Impact

API clients and automated tools may misinterpret endpoint capabilities. Breaks HTTP RFC compliance.

Location of Affected Code

API files in the back office and the frontend

Recommendation

Replace the default case in all API files

Team Response

Fixed.

[I-01] Unhandled Blockchain Transaction Errors Expose Low-Level Debug Data in UI

Severity

Informational Risk

Description

When certain operations in the backoffice interface fail (with wallet connected), such as initiating a transaction from the `Managed Pools` section under `Earn Products`, the application displays raw Ethereum transaction error details directly to the user.

The error includes:

- Function call data in hex
- The from and to addresses involved
- The exact contract method selector
- The error code (`CALL_EXCEPTION`)
- The library version (6.13.5)

This can be justified for being still in a sandbox environment or dev mode, but it's worth highlighting this issue due to its frequency in the Dapp. Example error:

```
Error: missing revert data (action="estimateGas", data=null, reason=null,
    transaction={ "data": "0xebd33e11...0000000", "from": "0x6682...", "
   to": "0x980bb5..." }, code=CALL_EXCEPTION, version=6.13.5)
```

This indicates the UI is exposing unprocessed exceptions from the blockchain call without user-friendly handling.

Justification by OWASP:

OWASP API Security Top 10 – API3:2023 – Broken Object Property Level Authorization While not a direct access control issue, the error exposes internal object structures (`transaction` , `data` , `code` , `version`) that should not be returned to frontend users. This undermines the principle of least privilege in data exposure.

OWASP API Security Top 10 – API6:2023 – Unrestricted Access to Sensitive Business Flows Detailed Ethereum transaction errors, including calldata and contract addresses, constitute sensitive backend information. Their exposure could allow attackers to reconstruct smart contract interactions, especially if function selectors are revealed.

OWASP API Security Top 10 – API8:2023 – Security Misconfiguration Allowing raw exceptions to bubble up to the frontend without sanitization is a clear example of misconfiguration. Developers should configure proper error handling and avoid leaking backend or blockchain-specific implementation details to untrusted clients.

Impact

Information Disclosure: Reveals internal contract method IDs and full calldata, which could be used to reverse-engineer contract logic or identify callable functions.

Location of Affected Code

Link: app.testnet.colb.io/app/balance

Link: backoffice.testnet.colb.io/

Recommendation

Client-Side Error Sanitization:

- Catch errors from blockchain calls before they propagate to the UI.
- Map common failure reasons (insufficient funds, contract revert, gas estimation failure) to user-friendly messages.

Team Response

Fixed.

[I-02] Undefined Error Handling in JWT Verification

Severity

Informational Risk

Description

JWT verification errors are silently swallowed, returning undefined.

```
catch (e) {
  return undefined; // Silent failure
}
```

Justification by OWASP:

OWASP ASVS 4.0 – V2: Authentication Verification V2.1.1: Verify that all authentication decisions are logged and that failures are handled securely, preventing silent bypasses.

Impact

Authentication failures may be ignored, allowing unauthorized access.

Location of Affected Code

File: apps/backoffice/middleware.ts

Recommendation

Throw and handle explicit errors on JWT verification failure.

Team Response

Fixed.

our shielding · Your smart contracts, our shielding · Your smart

shieldify

Thank you!