# DFINITY OISY

Security Assessment

**September 5, 2025**

*Prepared for:*
**Robin Künzler**
DFINITY

*Prepared by:* **Fredrik Dahlgren and Samuel Moelius**

# Table of Contents

## Project Summary

### Contact Information

The following project manager was associated with this project:

**Sam Greenup**, Project Manager
sam.greenup@trailofbits.com

The following engineering director was associated with this project:

**Jim Miller**, Engineering Director, Cryptography and Blockchain
james.miller@trailofbits.com

The following consultants were associated with this project:

**Fredrik Dahlgren**, Consultant
fredrik.dahlgren@trailofbits.com

**Samuel Moelius**, Consultant
samuel.moelius@trailofbits.com

### Project Timeline

The significant events and milestones of the project are listed below.

| Date | Event |
|---|---|
| **February 3, 2025** | Pre-project kickoff call |
| **March 13, 2025** | Status update meeting #1 |
| **March 31, 2025** | Delivery of report draft |
| **March 31, 2025** | Report readout meeting |
| **April 28, 2025** | Delivery of final comprehensive report |
| **September 5, 2025** | Completion of fix review |

# Executive Summary

## Engagement Overview

DFINITY engaged Trail of Bits to review the security of its OISY cryptocurrency wallet. The OISY wallet allows users to manage cryptocurrency assets from the browser, similar to MetaMask. However, authentication is handled using Internet Identity and keys are managed using the Internet Computer Chain Fusion Signer, which provides additional security over manually managed keys.

A team of two consultants conducted the review from March 4 to March 14, 2025, for a total of four engineer-weeks of effort. Our testing efforts focused on checking implementation correctness and security across the TypeScript front end and canister back ends. With full access to source code, documentation, and an OISY test deployment, we performed static and dynamic testing of the target, using automated and manual processes.

From September 3 to September 5, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the DFINITY team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue. For the results of this review, see appendix E.

## Observations and Impact

The OISY wallet consists of a web front end and a back end canister. The front end presents information to users and accepts requests that it passes on to the back end canister, Chain Fusion Signer, or third-party ledgers. Security vulnerabilities in the front end could involve misrepresentation of data to the user or mishandling of data passed on to the back end or third-party canisters. We found no major instances of either kind of vulnerability.

The back end canister receives and responds to user requests passed from the front end. Security vulnerabilities in the back end could involve mishandling of requests, missing validation checks, or incorrect execution of requests. We found no major instances of these types of vulnerabilities either.

Despite the above, we found minor cases where the project's security posture could be improved. For example, the unit test coverage across both the web front end and the back end canister is currently too low (TOB-OISY-13). The back end canister has additional integration tests that run under the `pocket-ic` test runner, but there is no automated way to determine test coverage for these tests, which makes it difficult to ascertain which components lack sufficient coverage. We also found that the security posture of the CI workflows could be improved to harden the CI/CD pipeline against attacks (TOB-OISY-5, TOB-OISY-6, TOB-OISY-7). A related area of improvement that we also want to highlight is the heavy reliance on shell scripts for testing and deployment, since these typically lack any form of input sanitization and increase the risk of command injection attacks.

## Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that DFINITY take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or any refactor that may occur when addressing other recommendations.

- **Invest in unit testing to improve measurable test coverage.** The unit test coverage across both the OISY front end and back end is currently too low (TOB-OISY-13). Many of the tests for the web front end are quite shallow and ensure only that a particular function was called without inspecting the actual state changes or effects of the call. The back end canister is covered by additional integration tests, but given that there is currently no way to compute test coverage for Wasm code, it is currently not possible to determine what parts of the back end are covered by the integration test suite. We recommend that DFINITY invest in more unit testing to improve measurable coverage of both the web front end and back end canister. In the long term, we also believe it would make sense to invest in a solution that adds coverage reporting functionality to `pocket-ic` or a similar tool.

- **Improve the security posture of the project's CI/CD pipeline.** The project contains a large number of workflows, many of which can be triggered by third parties using the `pull_request` trigger. Running `zizmor` on the GitHub workflows identifies a large number of potential areas for improvement across most of the workflows. (For more details, see TOB-OISY-5, TOB-OISY-6, and TOB-OISY-7.) Even though we did not find a way to compromise the CI/CD pipeline during the review, we still recommend that DFINITY take steps to further lock down the pipeline by restricting actions triggerable by third parties, locking down workflow permissions at the repository and organizational level, and fixing the issues highlighted by `zizmor`. We would also recommend that the project integrate either `zizmor` or a similar tool in the CI/CD pipeline to automatically identify potential security issues in GitHub workflows going forward.

- **Rewrite shell scripts in a language meant for application development.** The scripts directory contains 50 shell scripts, many of which produce `ShellCheck` warnings. Rewriting the scripts in a language meant for application development will make them more robust and will provide a more predictable experience for users.

- **Use automation to identify outdated and vulnerable dependencies.** Such automation could be the use of preexisting services (e.g., Dependabot) or a custom workflow. Adopting one of these strategies could prevent future issues like TOB-OISY-3 and TOB-OISY-4.

## Finding Severities and Categories

The following tables provide the number of findings by severity and category.

### EXPOSURE ANALYSIS

| *Severity* | *Count* |
|---|---|
| High | 0 |
| Medium | 1 |
| Low | 4 |
| Informational | 8 |
| Undetermined | 1 |

### CATEGORY BREAKDOWN

| *Category* | *Count* |
|---|---|
| Auditing and Logging | 1 |
| Configuration | 3 |
| Data Exposure | 1 |
| Data Validation | 2 |
| Patching | 4 |
| Testing | 3 |

## Project Targets

The engagement involved reviewing and testing the following target.

### oisy-wallet

| | |
|---|---|
| Repository | https://github.com/dfinity/oisy-wallet |
| Version | `57250d9b375189024f656382d23a4a675de06ee4` |
| Type | TypeScript, Rust |
| Platform | Web, Internet Computer |

## Project Goals

The engagement was scoped to provide a security assessment of the OISY wallet. Specifically, we sought to answer the following non-exhaustive list of questions:

- Does the OISY back end implement sufficient authentication and authorization checks?
- Are any of the back end canisters vulnerable to time-of-check to time-of-use vulnerabilities or other race conditions?
- Are errors handled and reported correctly by the back end?
- Does the front end default to using certified query calls or update calls when interacting with canisters?
- Does the front end use appropriate HTTP headers to mitigate common web-based vulnerabilities?
- Does the front end enforce HSTS?
- Is sensitive data like delegated identities stored securely in the browser?
- Is sensitive data removed from the browser when the user logs out?

# Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **Static analysis:** We ran Clippy and Semgrep over the Rust code and reviewed the results. We ran Semgrep and CodeQL over the TypeScript code and reviewed the results. Finally, we ran ShellCheck over the script files and reviewed the results.

- **Dependency analysis:**

  - **Rust dependencies:** We ran `cargo update` to look for compatible upgrades the project might be missing. We ran `cargo upgrade --incompatible` to look for incompatible upgrades the project might take advantage of. Finally, we ran `cargo audit` to look for dependencies with outstanding RustSec advisories.

  - **Node dependencies:** We ran `npm audit` to look for Node.js dependencies with known vulnerabilities.

- **Test coverage analysis:** We verified that all of the project's tests pass. We also computed the project's test coverage using `vitest run --coverage` and `cargo-llvm-cov` to look for important conditions the tests might have missed.

- **Back end canister:** We manually reviewed the back end canister and checked that each API endpoint implements the necessary validation checks and that the business logic and error handling is implemented correctly.

- **Wallet front end:** We performed a manual review of the OISY front end looking for correctness and interface issues that could cause the wallet to present misleading or invalid data to the user. We also checked that secrets are handled and stored properly by the wallet.

## Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- The review did not cover functionality related to the OISY rewards canister, as it was out of scope for the review.

- The review did not cover external packages like `@dfinity/auth-client`, `@dfinity/agent`, and `@dfinity/utils` used to authenticate and interact with the Internet Computer.

- When trying to test the wallet locally, we repeatedly encountered errors of the following form:

  ```
  The file does not exist at "..." which is in the optimize
  deps directory. The dependency might be incompatible with
  the dep optimizer. Try adding it to
  `optimizeDeps.exclude`.
  ```

  In most cases, rebuilding the wallet caused the errors to go away. Nonetheless, the errors slowed down our efforts to test the wallet locally. We were unable to determine the errors' root cause.

# Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in-house, to perform automated testing of source code and compiled software.

## Test Harness Configuration

We used the tools in the table below for the automated testing phase of this project. Instructions for installing the tools can be found in appendix E.

| Tool | Description |
|---|---|
| `cargo-audit` | A Cargo subcommand that can be used to audit project dependencies for known vulnerabilities |
| `cargo-llvm-cov` | A Cargo plugin for generating LLVM source–based code coverage |
| `cargo-upgrade` | A Cargo subcommand to upgrade dependencies in `Cargo.toml` to their latest versions |
| Clippy | A Rust linter used to catch common mistakes and unidiomatic Rust code |
| CodeQL | A static-analysis engine for variant analysis and vulnerability discovery |
| `npm audit` | An npm subcommand that identifies TypeScript dependencies with known vulnerabilities |
| Semgrep | An open-source static analysis tool for finding bugs and enforcing code standards when editing or committing code and during build time |
| ShellCheck | A tool that gives warnings and suggestions for Bash/sh shell scripts |
| Vitest | A JavaScript unit testing framework created to complement Vite, a tool for helping to manage and build JavaScript-based web applications |
| `zizmor` | A static analysis tool for GitHub actions |

## Areas of Focus

Our automated testing and verification work focused on the following:

- General code quality issues and unidiomatic code patterns
- Issues related to error handling and the use of `unwrap` and `expect`
- Poor unit and integration test coverage
- General issues with dependency management and known vulnerable dependencies
- Common mistakes and security issues in GitHub actions

## Test Results

The results of this focused testing are detailed below.

### `cargo-audit`

The `cargo-audit` tool identified a number of unmaintained dependencies. It also identified two project dependencies with outstanding RustSec advisories (TOB-OISY-4).

### `cargo-llvm-cov`

The coverage report obtained from `cargo-llvm-cov` shows test coverage only for the unit test suite. Since the integration tests run under the `pocket-ic` test runner, there is currently no way to collect test coverage reports for these tests.

The test coverage report for the unit test suite shows that the measurable test coverage could be improved for the back end canister (TOB-OISY-13).

### `cargo-upgrade`

Running `cargo upgrade --incompatible` revealed several incompatible upgrades, most of which were related to the Internet Computer (e.g., `ic-cdk`). The DFINITY team communicated that it is aware of the incompatible upgrades.

### Clippy

Running Clippy with the pedantic lints enabled produced only a few warnings regarding disallowed macros. Adding `--all-targets` produced several warnings regarding the tests. While we do not consider them to have security implications, we recommend addressing the warnings.

### CodeQL

Running CodeQL identified one informational-severity issue related to HTML parsing using regular expressions (TOB-OISY-10).

### `npm audit`

We ran `npm audit` on the project root directory. This identified a number of TypeScript dependencies with known vulnerabilities. However, none of the flagged vulnerabilities directly affect the OISY codebase.

### Semgrep

We ran Semgrep Pro on the codebase using both the default configuration and our internal rulesets. This analysis did not identify any security issues in the codebase.

### ShellCheck

We ran ShellCheck over the scripts in the `scripts` subdirectory, which produced a number of warnings (TOB-OISY-1)

### Vitest

Running Vitest showed that all front end tests pass. However, we also found that the test coverage for the front end could be improved (TOB-OISY-13).

### `zizmor`

Running `zizmor` to analyze the GitHub workflows used by the project identified two informational-severity issues (TOB-OISY-5, TOB-OISY-6) and one low-severity issue (TOB-OISY-7) with how GitHub actions are used.

## Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

| Category | Summary | Result |
|---|---|---|
| Arithmetic | Very little security-critical arithmetic is performed by the wallet front end or back end canisters. The front end uses the primitive `number` type to represent amounts, and we found no issues related to rounding or arithmetic overflows. | Satisfactory |
| Auditing | The back end canister performs little logging. However, it is adequate for the task the code is meant to address. | Satisfactory |
| Authentication / Access Controls | User authentication is implemented using Internet Identity, which provides a robust way to outsource OISY user authentication to an already existing, vetted protocol. The delegated session key is stored in the browser using IndexedDB, is not accessible to other tabs, and is deleted when the user logs out of the service. | Satisfactory |
| Complexity Management | The codebase is well structured and generally easy to navigate. Functional components are kept small, but some statements contain unnecessary nesting, which hurts readability.<br><br>A considerable amount of functionality related to testing and deployment is implemented as shell scripts. These are typically complex to maintain and increase the exposure to command injection attacks and the risk of upstream errors leading to unexpected behavior.<br><br>The CI/CD pipeline relies on a large number of custom actions and workflows, where multiple workflows can be triggered by third parties using the `pull_request` trigger. This increases the attack surface of the pipeline and the risk that a third party could compromise the repository by injecting attacker-controlled inputs into a | Moderate |

| | | |
|---|---|---|
| | workflow.<br><br>The back end implementation relies on dependencies with outstanding RustSec advisories. The back end test suite does not work with the latest version of the `pocket-ic` test runner. The back end codebase also contains obsolete code. | |
| Cryptography and Key Management | ICRC-1 signing keys are handled by the Internet Computer Chain Fusion Signer, which significantly reduces the exposure of the user's signing keys. The user interacts with the Chain Fusion Signer using a short-lived delegated session key obtained using the Internet Identity protocol.<br><br>However, we found that the authentication client implemented by `@dfinity/agent-js` will reuse session keys found in the browser's storage, which increases the risk of exposure of the private component of the session key (TOB-OISY-12). | **Satisfactory** |
| Decentralization | The client relies on a number of third-party REST APIs like Infura, Etherscan, and Blockchain.info to obtain transaction data. This constitutes a single point of failure, and a compromised third party could present an invalid view of the wallet state. | **Moderate** |
| Documentation | The code features extensive documentation describing the project's design and philosophy. | **Satisfactory** |
| Low-Level Manipulation | The codebase contains no `unsafe` code and does little bit- or byte-level manipulation. Hence, this category is not applicable. | **Not Applicable** |
| Testing and Verification | The unit test coverage across both the web front end and back end canister is too low, with 66% and 33% line coverage, respectively (TOB-OISY-13). The back end canister has additional integration tests, but since these run under `pocket-ic`, there is no way to determine what actual coverage they achieve. | **Moderate** |
| Transaction Ordering | We found no problems related to transaction ordering. | **Satisfactory** |

## Summary of Findings

The table below summarizes the findings of the review, including details on type and severity. For more information on the issues that the DFINITY team has addressed and on the mitigations implemented, see appendix E.

| ID | Title | Type | Severity |
|---|---|---|---|
| 1 | ShellCheck warnings | Patching | Informational |
| 2 | Tests require a script to run | Testing | Informational |
| 3 | Tests do not work with the latest version of pocket-ic | Testing | Informational |
| 4 | Reliance on vulnerable dependencies | Patching | Informational |
| 5 | Deployment workflow is vulnerable to cache poisoning | Configuration | Informational |
| 6 | Potential credential persistence in artifacts | Configuration | Informational |
| 7 | Unpinned external GitHub CI/CD action versions | Configuration | Low |
| 8 | The top_up_cycles_ledger API may return invalid data | Auditing and Logging | Low |
| 9 | Codebase contains obsolete code | Patching | Informational |
| 10 | HTML tags matched with regular expressions | Patching | Informational |
| 11 | Transaction IDs are not validated by the back end canister | Data Validation | Low |
| 12 | OISY reuses the identity key from browser storage on sign in | Data Exposure | Low |
| 13 | Measurable test coverage is low | Testing | Undetermined |

|  |  |  |  |
|---|---|---|---|
| 14 | Front end may misrepresent bitcoin transactions with multiple outputs | Data Validation | **Medium** |

# Detailed Findings

| 1. ShellCheck warnings | |
|---|---|
| Severity: **Informational** | Difficulty: **Not Applicable** |
| Type: Patching | Finding ID: TOB-OISY-1 |
| Target: `scripts` subdirectory | |

### Description

The `scripts` subdirectory contains 50 scripts, some of which are used for production purposes.[1] Running `shellcheck` against the scripts produces many warnings (table 1.1). The warnings should be addressed if the scripts are to continue to be used for production.

| Warning | Number of Occurrences |
|---|---|
| SC2001: See if you can use `${variable//search/replace}` instead. | 1 |
| SC2016: Expressions don't expand in single quotes, use double quotes for that. | 2 |
| SC2034: `foo` appears unused. Verify it or export it. | 5 |
| SC2038: Use `-print0/-0 or find -exec +` to allow for non-alphanumeric filenames. | 1 |
| SC2045: Iterating over `ls` output is fragile. Use globs. | 1 |
| SC2046: Quote this to prevent word splitting. | 5 |
| SC2059: Don't use variables in the `printf` format string. Use `printf "..%s.." "$foo"`. | 2 |
| SC2086: Double quote to prevent globbing and word splitting. | 15 |

*Table 1.1: Warnings produced by ShellCheck when run over the `scripts` subdirectory*

### Exploit Scenario

Eve finds a way to influence the contents of an unquoted variable in a shell script. Eve uses this capability to gain the ability to execute commands on Alice's machine.

[1] For example, the `package.json` file refers to six of these scripts.

### Recommendations

Short term, run `shellcheck` over the `scripts` subdirectory and address each of the warnings produced. Doing so will help ensure that running the scripts produces only the scripts' intended effects.

Long term, look for ways to rely less on shell scripts and more on languages built for application development. Doing so will reduce the risk to users needing the scripts' functionality.

### References

- ShellCheck: A shell script static analysis tool

| 2. Tests require a script to run | |
|---|---|
| Severity: **Informational** | Difficulty: **Not Applicable** |
| Type: Testing | Finding ID: TOB-OISY-2 |
| Target: `scripts/test.backend.sh` | |

## Description

The `backend` package's tests cannot be run simply with `cargo test`. Rather, a script is required to run them. Such a procedure reduces the likelihood that developers will run the tests, and it increases the attack surface for those that do.

An excerpt of the script appears in figure 2.1. The script contains 73 lines culminating in a call to `cargo test`. The lines preceding the call to `cargo test` include three invocations of `curl`, for example.

```
  1    #!/bin/bash
       ...
 24      curl -sSL
"https://github.com/dfinity/bitcoin-canister/releases/download/release%2F$BITCOIN_CA
NISTER_RELEASE/ic-btc-canister.wasm.gz" -o $BITCON_CANISTER_WASM
       ...
 37        curl -sSL
"https://github.com/dfinity/oisy-wallet/releases/download/${version}/backend.wasm.gz
" -o "$OISY_UPGRADE_PATH"
       ...
 60      curl -sSL
https://github.com/dfinity/pocketic/releases/download/${POCKET_IC_SERVER_VERSION}/po
cket-ic-x86_64-${PLATFORM}.gz -o ${POCKET_IC_SERVER_PATH}.gz
       ...
 70    # Run tests
 71
 72    echo "Running backend integration tests."
 73    cargo test -p backend "${@}"
```

*Figure 2.1: An excerpt of the script required to run the backend package's tests (`scripts/test.backend.sh#1–74`)*

## Exploit Scenario

Alice, a DFINITY developer, is unable to get the `scripts/test.backend.sh` script to work. She submits a change to the codebase without running the script. Alice introduces a bug into the codebase as a result.

**Recommendations**
Short term, eliminate the need for the `scripts/test.backend.sh` script. Take one of the following two steps:

- Package the needed assets as part of the package and prepare them in the package's build script.
- Perform the script's functionality directly from the tests that need them.

Taking either of these steps will make it easier for developers to run the tests, thereby increasing the likelihood that they are run.

Long term, as additional functionality is introduced into the codebase, avoid implementing it with shell scripts. Adding shell scripts to the codebase adds technical debt that later must be eliminated.

## 3. Tests do not work with the latest version of pocket-ic

| | |
|---|---|
| Severity: **Informational** | Difficulty: **Not Applicable** |
| Type: Testing | Finding ID: TOB-OISY-3 |
| Target: `src/backend/tests` | |

### Description

The `test.backend.sh` script uses an older version of `pocket-ic` (figure 3.1). Running the `backend` tests with the latest version of `pocket-ic` (8.0.0) produces the warning in figure 3.2. By using an older version of `pocket-ic`, the tests risk verifying that the code works with an older version of the Internet Computer, but not the current one.

```
#!/bin/bash

POCKET_IC_SERVER_VERSION=6.0.0
OISY_UPGRADE_VERSIONS="v0.0.13,v0.0.19"
BITCOIN_CANISTER_RELEASE="2024-08-30"
BITCON_CANISTER_WASM="ic-btc-canister.wasm.gz"
```

*Figure 3.1: Excerpt of `test.backend.sh` (`scripts/test.backend.sh#1–6`)*

```
error: unexpected argument '--pid' found

Usage: pocket-ic [OPTIONS]

For more information, try '--help'.
```

*Figure 3.2: Error produced when running the backend tests with the latest version of `pocket-ic`*

Note that the `--pid` option appears to have been removed in version 7.0.0.

### Exploit Scenario

Alice, a DFINITY developer, verifies that her code works by running the `test.backend.sh` script before submitting her changes to the repository. Unbeknownst to her, the Internet Computer has undergone changes not reflected in the version of `pocket-ic` that `test.backend.sh` uses. Alice introduces a bug in the repository as a result.

### Recommendations

Short term, update the `test.backend.sh` script to the latest version of `pocket-ic`. Doing so will help ensure that the OISY wallet works with the latest version of the Internet Computer.

Long term, consider using automation (e.g., a GitHub workflow) to identify when new versions of `pocket-ic` are available. Doing so will make it easier to keep the `test.backend.sh` script up to date.

| **4. Reliance on vulnerable dependencies** | |
|---|---|
| Severity: **Informational** | Difficulty: **Undetermined** |
| Type: Patching | Finding ID: TOB-OISY-4 |
| Target: `Cargo.lock` | |

## Description

The project relies on multiple dependencies with outstanding RustSec advisories (table 4.1). The dependencies should be updated so that the project does not rely on vulnerable code.

| Dependency | Level | RustSec advisory |
|---|---|---|
| `idna 0.5.0` | Error | `idna` accepts Punycode labels that do not produce any non-ASCII when decoded |
| `rustls 0.23.13` | Error | `rustls` network-reachable panic in `Acceptor::accept` |
| `ansi_term 0.12.1` | Warning | `ansi_term` is unmaintained |
| `atty 0.2.14` | Warning | `atty` is unmaintained |
| `dotenv 0.15.0` | Warning | `dotenv` is unmaintained |
| `proc-macro-error 1.0.4` | Warning | `proc-macro-error` is unmaintained |
| `serde_cbor 0.11.2` | Warning | `serde_cbor` is unmaintained |
| `atty 0.2.14` | Warning | Potential unaligned read |
| `bytes 1.6.0` | Warning | Yanked |
| `futures-util 0.3.30` | Warning | Yanked |

*Table 4.1: Warnings produced by `cargo audit` when run over the codebase*

Note that running `cargo update` resolves the two errors in table 4.1.

## Exploit Scenario

Eve discovers a vulnerability in a dependency that OISY uses and exploits it on OISY users' machines.

## Recommendations

Short term, run `cargo update` to resolve the two errors in table 4.1. Doing so will ensure that OISY users do not run the vulnerable code in the associated dependencies.

Long term, run `cargo audit` regularly, possibly with automation. Doing so will help to ensure that OISY does not rely on dependencies with outstanding RustSec advisories.

## 5. Deployment workflow is vulnerable to cache poisoning

| | |
|---|---|
| Severity: **Informational** | Difficulty: **High** |
| Type: Configuration | Finding ID: TOB-OISY-5 |
| Target: `.github/workflows/deploy-to-environment.yml` | |

### Description

The deployment workflow uses the GitHub cache for intermediate build artifacts. An attacker who gains write access to the GitHub actions cache (e.g., by leaking a valid GitHub actions runtime token) can poison the cache and compromise the build process to maliciously alter build artifacts.

```
- name: Restore cargo cache
  uses: actions/cache@v4
  with:
    path: |
      /home/runner/.cargo/registry
      /home/runner/.cargo/git
      target/
     ~/.cargo/bin/
  # [...]
```

*Figure 5.1: The deployment workflow uses the GitHub actions cache for intermediate build artifacts. (`.github/workflows/deploy-to-environment.yml#159–166`)*

### Exploit Scenario

An attacker gains unauthorized access to a GitHub token with write permissions to the GitHub actions cache. She uses this to poison the cache to inject a malicious project dependency into the local registry. When a new version is deployed, the malicious dependency is picked up by the build process, and a vulnerable version of OISY is deployed to the Internet Computer. Since the codebase is not tampered with directly, the attack goes unnoticed by the DFINITY team.

### Recommendations

Short term, in general, avoid using previously cached CI state within workflows intended to publish or deploy build artifacts such as canisters and Docker containers.

Long term, add `zizmor` to the CI/CD pipeline to detect GitHub action–related issues.

## 6. Potential credential persistence in artifacts

| | |
|---|---|
| Severity: **Informational** | Difficulty: **High** |
| Type: Configuration | Finding ID: TOB-OISY-6 |
| Target: `.github/workflows` | |

### Description

The default behavior of the `actions/checkout` GitHub action is to persist credentials to the locally checked out version of the repository. This allows subsequent actions to execute Git commands that require authentication. The credentials are stored inside the `.git` directory in the local root directory of the repository, where they could be inadvertently included in cached published build artifacts like release packages or Docker containers.

As an example, a workflow that caches individual layers of a Docker image to the GitHub actions cache and uses the command `COPY . .` to copy the entire repository into the container could leak any credentials stored inside the `.git` directory to the GitHub actions cache.

```
FROM deps AS build_frontend

COPY . .

ARG network="staging"
ENV ENV=$network
ENV DFX_NETWORK=$network

RUN echo $ENV
RUN echo $DFX_NETWORK

RUN npm ci
RUN npm run build
```

*Figure 6.1: The container used to build the OISY front end copies the entire repository, including the `.git` directory, into the container. (`Dockerfile.frontend#21–33`)*

### Recommendations

Short term, add `persist-credentials: false` to checkout actions that do not require privileged Git operations.

Long term, run `zizmor` as part of the CI/CD pipeline.

## 7. Unpinned external GitHub CI/CD action versions

| Severity: **Low** | Difficulty: **High** |
|---|---|
| Type: Configuration | Finding ID: TOB-OISY-7 |
| Target: `.github/workflows` | |

### Description

Several GitHub Actions workflows in the OISY repository use third-party actions pinned to a tag or branch name instead of a full-length commit SHA, as recommended by GitHub. This configuration enables repository owners to silently modify the actions. A malicious actor could use this ability to tamper with release artifacts or leak repository secrets.

The following actions are owned by GitHub organizations or individuals that are not affiliated directly with DFINITY:

- `peter-evans/create-pull-request@v6`
- `docker/setup-buildx-action@v3`
- `docker/build-push-action@v5`
- `release-drafter/release-drafter@v6`
- `EndBug/add-and-commit@v9`
- `dorny/paths-filter@v3`

```
- name: Commit and push updates
  uses: EndBug/add-and-commit@v9
  if: env.FINAL_CHANGES == 'true' && github.ref != 'refs/heads/main'
  with:
    add: e2e
    default_author: github_actions
    message: '🤖 Update E2E snapshots'
```

*Figure 7.1: GitHub workflows depend on a number of third-party actions that are pinned only to a tag. (`.github/workflows/ci.yml#L27–L31`)*

### Exploit Scenario

An attacker gains unauthorized access to the account of a GitHub action owner. The attacker manipulates the action's code to steal GitHub credentials and compromise the repository.

### Recommendations

Short term, pin each third-party action to a specific full-length commit SHA, as recommended by GitHub. Additionally, configure Dependabot to update the actions' commit SHAs after reviewing their available updates.

Long term, add `zizmor` to the CI/CD pipeline.

| 8. The top_up_cycles_ledger API may return invalid data | |
|---|---|
| Severity: **Low** | Difficulty: **Low** |
| Type: Auditing and Logging | Finding ID: TOB-OISY-8 |
| Target: `src/backend/src/signer.rs`, `src/shared/src/types.rs` | |

## Description

The `top_up_cycles_ledger` API endpoint returns incorrect balance information when no top-up is needed. Specifically, the API returns 0 for the `ledger_balance` field while using the actual ledger balance for the `backend_cycles` field. This creates inconsistent and misleading state reporting that could cause issues for clients relying on this information for accounting or decision-making purposes.

The issue occurs in the return statement in the case when no top-up is needed.

```
Ok(TopUpCyclesLedgerResponse {
    ledger_balance: Nat::from(0u32),
    backend_cycles: ledger_balance,
    topped_up: Nat::from(0u32),
})
```

*Figure 8.1: The `ledger_balance` and `backend_cycles` fields are assigned incorrect values.* (*`src/backend/src/signer.rs#205–209`*)

In this case, the `ledger_balance` value should be equal to the balance returned by the cycles canister, and the `backend_cycles` value should be the back end canister cycle balance given by `Nat::from(ic_cdk::api::canister_balance128())`.

## Exploit Scenario

An automatic accounting system relies on the `top_up_cycles_ledger` API to track cycle balances across the system. When the ledger does not need a top-up, the system receives incorrect balance information, showing 0 for the ledger balance when it actually has funds. The accounting system triggers unnecessary alerts and top-up requests, causing operational overhead and potential system instability.

## Recommendations

Short term, correct the values in the return statement to correctly reflect the actual state: `ledger_balance` should be assigned to the `ledger_balance` field, and the current number of back end cycles should be assigned to the `backend_cycles` field.

Long term, implement comprehensive unit tests that verify the response structure and values for all possible scenarios, including the no-top-up case.

| 9. Codebase contains obsolete code | |
|---|---|
| Severity: **Informational** | Difficulty: **Not Applicable** |
| Type: Patching | Finding ID: TOB-OISY-9 |
| Target: backend canisters | |

## Description

The backend canisters contain migration code that is no longer needed. An example appears in figure 9.1. Obsolete code is rarely given the same level of scrutiny as code intended for continued use. Thus, the code should be removed.

```
pub fn may_write_user_data() -> Result<(), String> {
    caller_is_not_anonymous()?;
    if read_config(|s| s.api.unwrap_or_default().user_data.writable()) {
        Ok(())
    } else {
        Err("User data is in read only mode due to a migration.".to_string())
    }
}
```

*Figure 9.1: Migration code that is no longer needed*
*(`oisy-wallet/src/backend/src/guards.rs#24–31`)*

## Exploit Scenario

Alice, a DFINITY developer, writes code that calls into the obsolete migration code. Alice's changes make it more difficult for Bob to remove the migration code, because existing, in-production code refers to it.

## Recommendations

Short term, remove the obsolete migration code. Doing so will help prevent accidental continued use of the code.

Long term, track obsolete code in GitHub issues or some other bug tracking software. Doing so will help ensure the code is not forgotten.

## 10. HTML tags matched with regular expressions

| | |
|---|---|
| Severity: **Informational** | Difficulty: **High** |
| Type: Patching | Finding ID: TOB-OISY-10 |
| Target: `scripts` subdirectory | |

### Description

Several scripts within the `scripts` subdirectory use regular expressions to match HTML tags. Such an approach is difficult to get right. For example, the highlighted regular expression in figure 10.1 from the `build.csp.mjs` script does not match capitalized HTML tags. To help ensure the scripts' longevity, a proper HTML parser should be used.

```
130    const sw = /<script[\s\S]*?>([\s\S]*?)<\/script>/gm;
```

*Figure 10.1: Regular expression used to match HTML tags in the `build.csp.mjs` script (`oisy-wallet/scripts/build.csp.mjs#130`)*

### Exploit Scenario

Wiley, a Svelte developer, decides the code that Svelte produces would look cleaner if all tags were capitalized. Wiley pushes this change. OISY's `build.csp.mjs` script fails to identify the <SCRIPT> tags and generates improper `frame-src` headers. The OISY wallet ceases to work correctly as a result.

### Recommendations

Short term, eliminate all uses of regular expressions applied to HTML in the `scripts` subdirectory. Instead, use an HTML parser to extract the needed content. A parser is more likely to endure changes to the HTML produced by subordinate tools, such as Svelte.

Long term, consider running CodeQL regularly over the codebase. The example in figure 10.1 was flagged by CodeQL's `BadTagFilter.ql` query.

### References

- CodeQL: Bad HTML filtering regexp

## 11. Transaction IDs are not validated by the back end canister

| | |
|---|---|
| Severity: **Low** | Difficulty: **Medium** |
| Type: Data Validation | Finding ID: TOB-OISY-11 |
| Target: `src/backend/src/lib.rs` | |

### Description

The back end canister's `btc_add_pending_transaction` endpoint accepts transaction IDs without any size validation before storing them in the pending transactions storage. The code directly assigns the user-provided `txid` field from the request parameters to the storage structure without performing any bounds checking or format validation. This could allow an attacker to store arbitrarily large data in the canister's state, potentially exhausting its heap storage capacity.

```
#[update(guard = "may_write_user_data")]
pub async fn btc_add_pending_transaction(
    params: BtcAddPendingTransactionRequest,
) -> Result<(), BtcAddPendingTransactionError> {
    // ...

    with_btc_pending_transactions(|pending_transactions| {
        pending_transactions
            .prune_pending_transactions(principal, &current_utxos, now_ns);
        let current_pending_transaction = StoredPendingTransaction {
            txid: params.txid,
            utxos: params.utxos,
            created_at_timestamp_ns: now_ns,
        };
        pending_transactions
            .add_pending_transaction(
                principal,
                params.address,
                current_pending_transaction
            )
            .map_err(|msg| BtcAddPendingTransactionError::InternalError { msg })
    })
}
```

*Figure 11.1: The transaction ID from the user-supplied `params` argument is stored directly without validation. (`src/backend/src/lib.rs#414–439`)*

Each UTXO in the `params.utxos` vector also contains a transaction ID that should be checked to ensure that it has the expected size.

## Exploit Scenario

An attacker exploits the lack of transaction ID validation to perform a resource exhaustion attack against the OISY back end canister. The attacker writes a script that repeatedly calls `btc_add_pending_transaction` with very large byte vectors (e.g., 2 MiB) as transaction IDs. Since these transaction IDs are persisted by the canister, the attacker can quickly fill up the canister's heap, preventing tracking of legitimate users' transactions.

## Recommendations

Short term, implement size validation for transaction IDs before they are stored. Bitcoin transaction IDs are 32-byte hashes, so add a check to ensure that user-supplied transaction IDs are exactly 32 bytes long.

Long term, implement a comprehensive input validation framework that enforces strict bounds and format checking for all incoming, user-controlled parameters. Consider adding runtime metrics to monitor the amount of heap memory used by each canister.

## 12. OISY reuses the identity key from browser storage on sign in

| | |
|---|---|
| Severity: **Low** | Difficulty: **High** |
| Type: Data Exposure | Finding ID: TOB-OISY-12 |
| Target: `agent-js/packages/auth-client/src/index.ts` | |

### Description

When the OISY wallet signs out, the `AuthClient` instance will remove the delegated identity from the browser's IndexedDB. The wallet will then reload the page, which calls `syncAuthStore()`. This creates a new `AuthClient` by calling `createAuthClient`, which immediately creates a new ECDSA identity key and stores this key in the IndexedDB.

```
sync: async () => {
  authClient = authClient ?? (await createAuthClient());
  const isAuthenticated: boolean = await authClient.isAuthenticated();

  set({
    identity: isAuthenticated ? authClient.getIdentity() : null
  });
},
```

*Figure 12.1: When the page is reloaded and the `AuthStore` is synchronized, a new `AuthClient` instance is created. (`src/frontend/src/lib/stores/auth.store.ts#42–49`)*

```
if (!key) {
  // Create a new key (whether or not one was in storage).
  if (keyType === ED25519_KEY_LABEL) {
    key = await Ed25519KeyIdentity.generate();
    await storage.set(
        KEY_STORAGE_KEY, JSON.stringify((key as Ed25519KeyIdentity).toJSON()));
  } else {
    if (options.storage && keyType === ECDSA_KEY_LABEL) {
      console.warn(`Long warning removed...`);
    }
    key = await ECDSAKeyIdentity.generate();
    await storage.set(KEY_STORAGE_KEY, (key as ECDSAKeyIdentity).getKeyPair());
  }
}
```

*Figure 12.2: The new `AuthClient` instance stores a new identity key in IndexedDB in `AuthClient.create`. (`agent-js/packages/auth-client/src/index.ts#325–339`)*

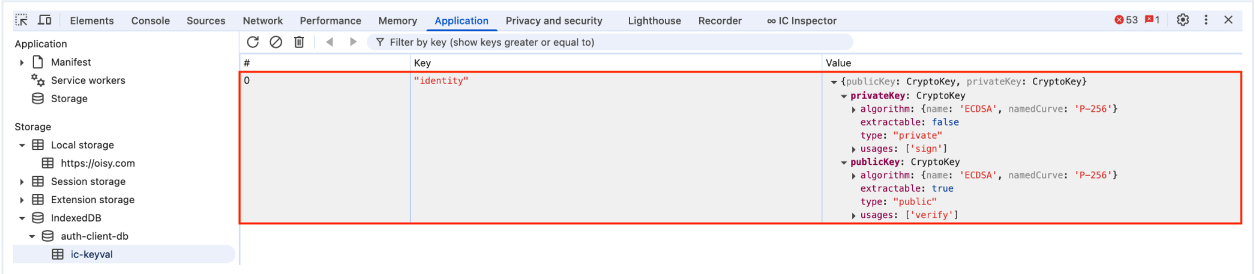
This behavior, while unexpected, does not constitute a security vulnerability in itself. However, the next time a user logs in, the `AuthClient` will read the existing identity key from the IndexedDB and use this key with Internet Identity to obtain a signed delegation.

```
let maybeIdentityStorage = await storage.get(KEY_STORAGE_KEY);
// ...

if (maybeIdentityStorage) {
  try {
    if (typeof maybeIdentityStorage === 'object') {
      if (keyType === ED25519_KEY_LABEL
          && typeof maybeIdentityStorage === 'string') {
        key = await Ed25519KeyIdentity.fromJSON(maybeIdentityStorage);
      } else {
        key = await ECDSAKeyIdentity.fromKeyPair(maybeIdentityStorage);
      }
    } else if (typeof maybeIdentityStorage === 'string') {
      // This is a legacy identity, which is a serialized Ed25519KeyIdentity.
      key = Ed25519KeyIdentity.fromJSON(maybeIdentityStorage);
    }
  } catch {
    // Ignore this, this means that the localStorage value isn't a valid
    // Ed25519KeyIdentity or ECDSAKeyIdentity serialization.
  }
}
```

*Figure 12.3: The new `AuthClient` instance will use the previously generated session key found in the IndexedDB. (`agent-js/packages/auth-client/src/index.ts#238–276`)*

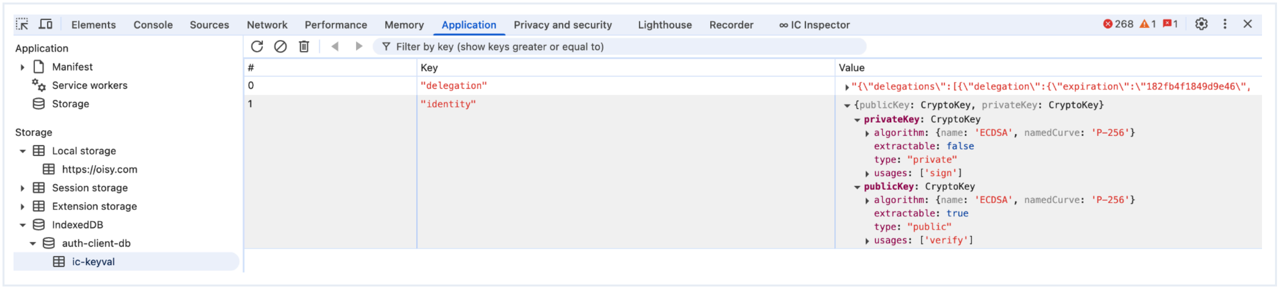
This means that an attacker with access to the browser could choose the identity key for the next logged-in user. However, without obtaining the signed delegation, there is no way for the attacker to forge messages on behalf of the user.



| # | Key | Value |
|---|---|---|
| 0 | "identity" | {publicKey: CryptoKey, privateKey: CryptoKey} |

```
▾ {publicKey: CryptoKey, privateKey: CryptoKey}
  ▾ privateKey: CryptoKey
    ▸ algorithm: {name: 'ECDSA', namedCurve: 'P-256'}
      extractable: false
      type: "private"
    ▸ usages: ['sign']
  ▾ publicKey: CryptoKey
    ▸ algorithm: {name: 'ECDSA', namedCurve: 'P-256'}
      extractable: true
      type: "public"
    ▸ usages: ['verify']
```

*Figure 12.4: The client generates a new identity key when the user signs out. This identity key is left in the IndexedDB object store.*



| # | Key | Value |
|---|---|---|
| 0 | "delegation" | "{\"delegations\":[{\"delegation\":{\"expiration\":\"182fb4f1849d9e46\", |
| 1 | "identity" | {publicKey: CryptoKey, privateKey: CryptoKey} |

```
{publicKey: CryptoKey, privateKey: CryptoKey}
  privateKey: CryptoKey
    algorithm: {name: 'ECDSA', namedCurve: 'P-256'}
    extractable: false
    type: "private"
    usages: ['sign']
  publicKey: CryptoKey
    algorithm: {name: 'ECDSA', namedCurve: 'P-256'}
    extractable: true
    type: "public"
    usages: ['verify']
```

*Figure 12.5: The next time the client signs in, it obtains a signed delegation on the previously generated key found in the browser's IndexedDB.*

The authentication client protects the identity key in two ways: when the key is saved to the IndexedDB, the client sets the `exportable` property on the private key to false to ensure that the key cannot be read from JavaScript. Furthermore, the client deletes the identity key when the user signs out, to limit the exposure of the private key. This issue allows an attacker to bypass both of these protections since the attacker can choose the identity key freely *before* the user signs in.

## Exploit Scenario

Mallory, a malicious user, stores a known key pair as the identity key in the authentication client object store of Alice's browser. When Alice signs in to OISY, Malory is able to copy the signed delegation corresponding to the key since it is not protected by the browser. Malory can now forge OISY requests from Alice and is able to steal any tokens owned by her.

## Recommendations

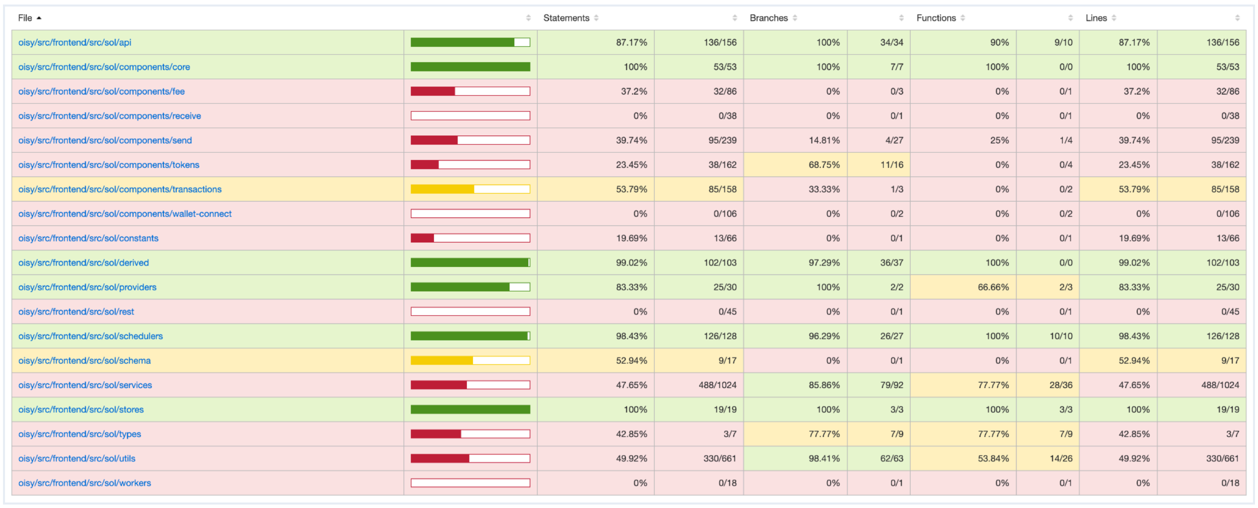
Short term, rewrite the synchronization code to prevent a new identity key from being generated when the user signs out, and ensure that a fresh identity key is generated when the user logs in.

Long term, add unit tests checking that the authentication client object store is cleared when the user signs out.

| 13. Measurable test coverage is low | |
|---|---|
| Severity: **Undetermined** | Difficulty: **High** |
| Type: Testing | Finding ID: TOB-OISY-13 |
| Target: `dfinity/oisy-wallet` repository | |

## Description

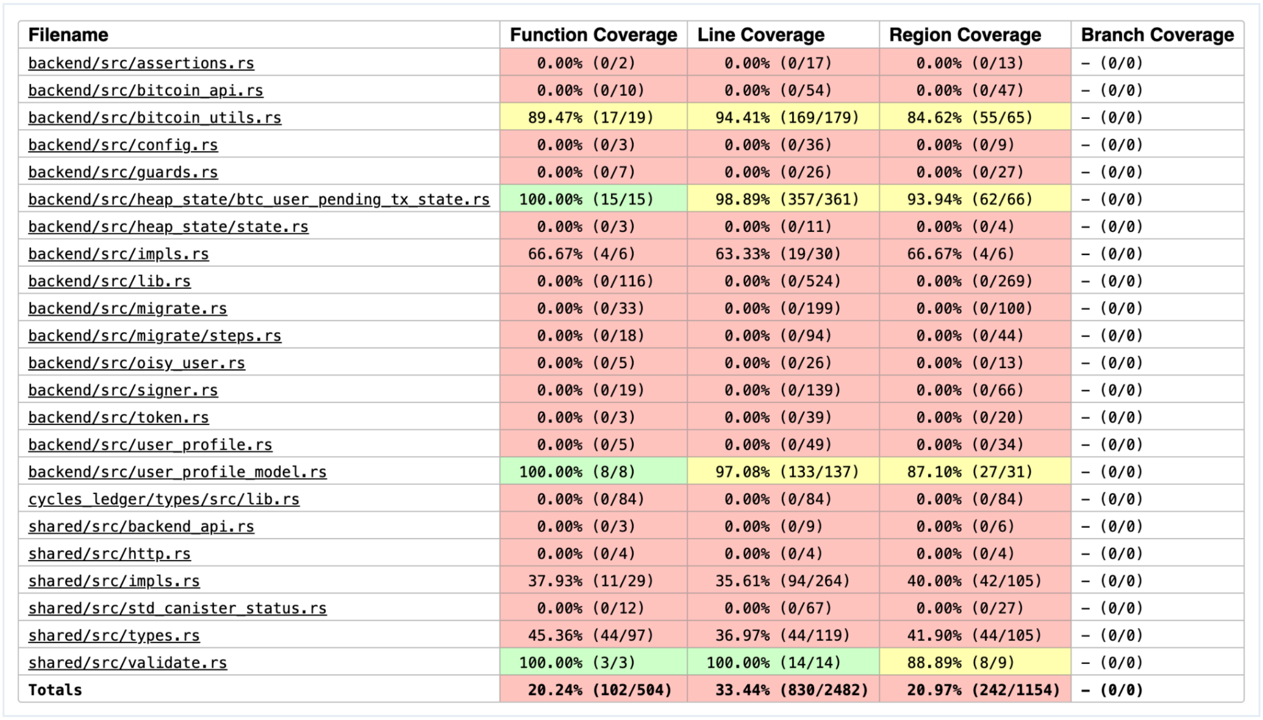
Running `npm run test:coverage` shows that the unit test coverage across the front end is too low. The average line coverage across the TypeScript code in the project is around 66%. Furthermore, many of the tests in the test suite only verify that a particular simple interaction took place (e.g., that a particular function is called when a button is clicked), but fail to validate state changes or other effects.



| File | | Statements | | Branches | | Functions | | Lines | |
|---|---|---|---|---|---|---|---|---|---|
| oisy/src/frontend/src/sol/api | | 87.17% | 136/156 | 100% | 34/34 | 90% | 9/10 | 87.17% | 136/156 |
| oisy/src/frontend/src/sol/components/core | | 100% | 53/53 | 100% | 7/7 | 100% | 0/0 | 100% | 53/53 |
| oisy/src/frontend/src/sol/components/fee | | 37.2% | 32/86 | 0% | 0/3 | 0% | 0/1 | 37.2% | 32/86 |
| oisy/src/frontend/src/sol/components/receive | | 0% | 0/38 | 0% | 0/1 | 0% | 0/1 | 0% | 0/38 |
| oisy/src/frontend/src/sol/components/send | | 39.74% | 95/239 | 14.81% | 4/27 | 25% | 1/4 | 39.74% | 95/239 |
| oisy/src/frontend/src/sol/components/tokens | | 23.45% | 38/162 | 68.75% | 11/16 | 0% | 0/4 | 23.45% | 38/162 |
| oisy/src/frontend/src/sol/components/transactions | | 53.79% | 85/158 | 33.33% | 1/3 | 0% | 0/2 | 53.79% | 85/158 |
| oisy/src/frontend/src/sol/components/wallet-connect | | 0% | 0/106 | 0% | 0/2 | 0% | 0/2 | 0% | 0/106 |
| oisy/src/frontend/src/sol/constants | | 19.69% | 13/66 | 0% | 0/1 | 0% | 0/1 | 19.69% | 13/66 |
| oisy/src/frontend/src/sol/derived | | 99.02% | 102/103 | 97.29% | 36/37 | 100% | 0/0 | 99.02% | 102/103 |
| oisy/src/frontend/src/sol/providers | | 83.33% | 25/30 | 100% | 2/2 | 66.66% | 2/3 | 83.33% | 25/30 |
| oisy/src/frontend/src/sol/rest | | 0% | 0/45 | 0% | 0/1 | 0% | 0/1 | 0% | 0/45 |
| oisy/src/frontend/src/sol/schedulers | | 98.43% | 126/128 | 96.29% | 26/27 | 100% | 10/10 | 98.43% | 126/128 |
| oisy/src/frontend/src/sol/schema | | 52.94% | 9/17 | 0% | 0/1 | 0% | 0/1 | 52.94% | 9/17 |
| oisy/src/frontend/src/sol/services | | 47.65% | 488/1024 | 85.86% | 79/92 | 77.77% | 28/36 | 47.65% | 488/1024 |
| oisy/src/frontend/src/sol/stores | | 100% | 19/19 | 100% | 3/3 | 100% | 3/3 | 100% | 19/19 |
| oisy/src/frontend/src/sol/types | | 42.85% | 3/7 | 77.77% | 7/9 | 77.77% | 7/9 | 42.85% | 3/7 |
| oisy/src/frontend/src/sol/utils | | 49.92% | 330/661 | 98.41% | 62/63 | 53.84% | 14/26 | 49.92% | 330/661 |
| oisy/src/frontend/src/sol/workers | | 0% | 0/18 | 0% | 0/1 | 0% | 0/1 | 0% | 0/18 |

*Figure 13.1: Test coverage for the OISY Solana integration*

We also generated test coverage reports for the back end canister using `cargo-llvm-cov`. This report also indicates that test coverage is low, with around 33% average line coverage across the back end canisters. However, since the integration test suite uses the `pocket-ic` test runner, there is currently no way to automatically measure test coverage for the integration tests.



| Filename | Function Coverage | Line Coverage | Region Coverage | Branch Coverage |
|---|---|---|---|---|
| backend/src/assertions.rs | 0.00% (0/2) | 0.00% (0/17) | 0.00% (0/13) | - (0/0) |
| backend/src/bitcoin_api.rs | 0.00% (0/10) | 0.00% (0/54) | 0.00% (0/47) | - (0/0) |
| backend/src/bitcoin_utils.rs | 89.47% (17/19) | 94.41% (169/179) | 84.62% (55/65) | - (0/0) |
| backend/src/config.rs | 0.00% (0/3) | 0.00% (0/36) | 0.00% (0/9) | - (0/0) |
| backend/src/guards.rs | 0.00% (0/7) | 0.00% (0/26) | 0.00% (0/27) | - (0/0) |
| backend/src/heap_state/btc_user_pending_tx_state.rs | 100.00% (15/15) | 98.89% (357/361) | 93.94% (62/66) | - (0/0) |
| backend/src/heap_state/state.rs | 0.00% (0/3) | 0.00% (0/11) | 0.00% (0/4) | - (0/0) |
| backend/src/impls.rs | 66.67% (4/6) | 63.33% (19/30) | 66.67% (4/6) | - (0/0) |
| backend/src/lib.rs | 0.00% (0/116) | 0.00% (0/524) | 0.00% (0/269) | - (0/0) |
| backend/src/migrate.rs | 0.00% (0/33) | 0.00% (0/199) | 0.00% (0/100) | - (0/0) |
| backend/src/migrate/steps.rs | 0.00% (0/18) | 0.00% (0/94) | 0.00% (0/44) | - (0/0) |
| backend/src/oisy_user.rs | 0.00% (0/5) | 0.00% (0/26) | 0.00% (0/13) | - (0/0) |
| backend/src/signer.rs | 0.00% (0/19) | 0.00% (0/139) | 0.00% (0/66) | - (0/0) |
| backend/src/token.rs | 0.00% (0/3) | 0.00% (0/39) | 0.00% (0/20) | - (0/0) |
| backend/src/user_profile.rs | 0.00% (0/5) | 0.00% (0/49) | 0.00% (0/34) | - (0/0) |
| backend/src/user_profile_model.rs | 100.00% (8/8) | 97.08% (133/137) | 87.10% (27/31) | - (0/0) |
| cycles_ledger/types/src/lib.rs | 0.00% (0/84) | 0.00% (0/84) | 0.00% (0/84) | - (0/0) |
| shared/src/backend_api.rs | 0.00% (0/3) | 0.00% (0/9) | 0.00% (0/6) | - (0/0) |
| shared/src/http.rs | 0.00% (0/4) | 0.00% (0/4) | 0.00% (0/4) | - (0/0) |
| shared/src/impls.rs | 37.93% (11/29) | 35.61% (94/264) | 40.00% (42/105) | - (0/0) |
| shared/src/std_canister_status.rs | 0.00% (0/12) | 0.00% (0/67) | 0.00% (0/27) | - (0/0) |
| shared/src/types.rs | 45.36% (44/97) | 36.97% (44/119) | 41.90% (44/105) | - (0/0) |
| shared/src/validate.rs | 100.00% (3/3) | 100.00% (14/14) | 88.89% (8/9) | - (0/0) |
| **Totals** | **20.24% (102/504)** | **33.44% (830/2482)** | **20.97% (242/1154)** | **- (0/0)** |

*Figure 13.2: Measurable test coverage for the back end canister*

The actual test coverage across both the unit and integration tests is most likely higher than indicated by the report produced by `cargo-llvm-cov`. However, due to the inability to accurately measure integration test coverage, it is unclear which components lack test coverage and where to invest in more testing. Additionally, ensuring that all the important functions and critical code paths are covered by the test suite would require extensive manual work.

### Exploit Scenario
An update to the back end canister inadvertently introduces a bug. Since the code is not covered by the unit test suite, the implementation remains untested and the issue remains undetected. When the new version of the back end is deployed to production, the issue manifests itself, causing reputational damage for the project.

### Recommendations
Short term, work toward improving measurable test coverage for both TypeScript and Rust code.

Long term, introduce randomized testing to further exercise the implementation on unexpected and invalid inputs.

| 14. Front end may misrepresent bitcoin transactions with multiple outputs | |
|---|---|
| Severity: **Medium** | Difficulty: **Low** |
| Type: Data Validation | Finding ID: TOB-OISY-14 |
| Target: `src/frontend/src/btc/utils/btc-transactions.utils.ts` | |

## Description

Bitcoin transactions typically have multiple outputs, each of which may have a unique recipient. However, when a bitcoin transaction is displayed by the front end, it will show only the value and recipient address for the first output.

Recent bitcoin transactions are fetched from the `blockchain.info` REST API by the `btcAddressData` function. The returned transactions are converted to `BtcTransactionUi` instances by the `mapBtcTransaction` function.

```
const { totalOutputValue, value, to } = out.reduce<{
  totalOutputValue: number;
  value: number | undefined;
  to: string | undefined;
}>(
  (acc, { addr, value }) => {
  // TODO: test what happens when user sends to the current address
  const isValidOutput =
    (isTypeSend && addr !== btcAddress) || (!isTypeSend && addr === btcAddress);

    if (isNullish(acc.value) && isValidOutput) {
      acc.value = (acc.value ?? 0) + value;
    }

    if (isNullish(acc.to) && isValidOutput) {
      acc.to = addr;
    }

    return {
      ...acc,
      totalOutputValue: acc.totalOutputValue + value
    };
  },
  {
    totalOutputValue: 0,
    value: undefined,
    to: undefined
  }
);
```

```
// ...

return {
  id: hash,
  timestamp: BigInt(time),
  value: nonNullish(value)
    ? BigInt(isTypeSend ? value + utxosFee : value)
    : undefined,
  status,
  blockNumber: block_index ?? undefined,
  type: isTypeSend ? 'send' : 'receive',
  from: isTypeSend ? btcAddress : inputs[0].prev_out.addr,
  to,
  confirmations
};
```

*Figure 14.1: The `mapBtcTransaction` function drops all transaction outputs except the first one. (`src/frontend/src/btc/utils/btc-transactions.utils.ts#31–59`)*

This function takes each transaction and creates a new `BtcTransactionUi` instance with the value and recipient equal to the value and recipient of the first transaction output. Any remaining outputs are ignored by the function, which means that the transaction data displayed by the OISY front end may be incomplete. This could cause confusion for the user since the data presented by the front end will not reflect the actual transaction data in general.

## Exploit Scenario

Alice, an OISY wallet user, submits a bitcoin transaction with multiple recipients to the Bitcoin network. When the transaction is displayed by the OISY wallet, it only shows the first transaction output. Alice submits a second transaction with the remaining outputs, causing her to double-spend the corresponding funds.

## Recommendations

Short term, update the OISY wallet UI to show all bitcoin transaction outputs.

Long term, document functions that parse incoming blockchain data to keep track of which transaction types are supported by the implementation. Add unit tests to ensure that transaction data is parsed correctly.

## A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

| Vulnerability Categories | |
|---|---|
| **Category** | **Description** |
| **Access Controls** | Insufficient authorization or assessment of rights |
| **Auditing and Logging** | Insufficient auditing of actions or logging of problems |
| **Authentication** | Improper identification of users |
| **Configuration** | Misconfigured servers, devices, or software components |
| **Cryptography** | A breach of system confidentiality or integrity |
| **Data Exposure** | Exposure of sensitive information |
| **Data Validation** | Improper reliance on the structure or values of data |
| **Denial of Service** | A system failure with an availability impact |
| **Error Reporting** | Insecure or insufficient reporting of error conditions |
| **Patching** | Use of an outdated software package or library |
| **Session Management** | Improper identification of authenticated users |
| **Testing** | Insufficient test methodology or test coverage |
| **Timing** | Race conditions or other order-of-operations flaws |
| **Undefined Behavior** | Undefined behavior triggered within the system |

| Severity Levels | |
|---|---|
| **Severity** | **Description** |
| Informational | The issue does not pose an immediate risk but is relevant to security best practices. |
| Undetermined | The extent of the risk was not determined during this engagement. |
| Low | The risk is small or is not one the client has indicated is important. |
| Medium | User information is at risk; exploitation could pose reputational, legal, or moderate financial risks. |
| High | The flaw could affect numerous users and have serious reputational, legal, or financial implications. |

| Difficulty Levels | |
|---|---|
| **Difficulty** | **Description** |
| Undetermined | The difficulty of exploitation was not determined during this engagement. |
| Low | The flaw is well known; public tools for its exploitation exist or can be scripted. |
| Medium | An attacker must write an exploit or will need in-depth knowledge of the system. |
| High | An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue. |

## B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories

| Category | Description |
|---|---|
| **Arithmetic** | The proper use of mathematical operations and semantics |
| **Auditing** | The use of event auditing and logging to support monitoring |
| **Authentication / Access Controls** | The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system |
| **Complexity Management** | The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions |
| **Cryptography and Key Management** | The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution |
| **Decentralization** | The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades |
| **Documentation** | The presence of comprehensive and readable codebase documentation |
| **Low-Level Manipulation** | The justified use of inline assembly and low-level calls |
| **Testing and Verification** | The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage |
| **Transaction Ordering** | The system’s resistance to transaction-ordering attacks |

| Rating Criteria | |
|---|---|
| **Rating** | **Description** |
| **Strong** | No issues were found, and the system exceeds industry standards. |
| **Satisfactory** | Minor issues were found, but the system is compliant with best practices. |
| **Moderate** | Some issues that may affect system safety were found. |
| **Weak** | Many issues that affect system safety were found. |
| **Missing** | A required component is missing, significantly affecting system safety. |
| **Not Applicable** | The category does not apply to this review. |
| **Not Considered** | The category was not considered in this review. |
| **Further Investigation Required** | Further investigation is required to reach a meaningful conclusion. |

# C. Automated Testing

This section describes the setup of the automated analysis tools used during this audit.

## cargo-audit

The `cargo-audit` Cargo plugin identifies known vulnerable dependencies in Rust projects. It can be installed using `cargo install cargo-audit`. To run the tool, run `cargo audit` in the repository's root directory.

## cargo-llvm-cov

The `cargo-llvm-cov` Cargo plugin is used to generate LLVM source–based code coverage data. The plugin can be installed via the command `cargo install cargo-llvm-cov`. To run the tool, run the command `cargo llvm-cov` in the repository's root directory.

## cargo-upgrade

The `cargo-upgrade` Cargo plugin identifies upgradeable dependencies in a project's `Cargo.toml` file. The plugin is part of the `cargo-edit` tool suite and can be installed via the command `cargo install cargo-edit`. To identify incompatible upgrades (i.e., upgrades that `cargo update` would not perform), run the command `cargo upgrade --incompatible` in the repository's root directory.

## Clippy

The Rust linter Clippy can be installed using `rustup` by running the command `rustup component add clippy`. Invoking `cargo clippy` in the root directory of the project runs the tool.

## CodeQL

The Trail of Bits Testing Handbook provides instructions for installing CodeQL either manually or with the Homebrew package manager. To reproduce the results in TOB-OISY-10, one must construct a CodeQL database and run the `BadTagFilter.ql` query, using the following commands:

```
codeql database create codeql.db --language=javascript

codeql database analyze codeql.db --format=sarif-latest
--output=results.sarif -- codeql/javascript-queries
```

## npm audit

The `audit` subcommand is built into the Node Package Manager (npm). The subcommand is run and a summary is printed whenever `npm install` is run. More detailed output can be obtained by running `npm audit` directly.

## Semgrep

Semgrep can be installed using `pip` by running `python3 -m pip install semgrep`. To run Semgrep on a codebase, run `semgrep --config "<CONFIGURATION>"` in the root directory of the project. Here, `<CONFIGURATION>` can be a single rule, a directory of rules, or the name of a ruleset hosted on the Semgrep registry. To use the default configuration, use the command-line argument `--config auto`. Trail of Bits' public ruleset can be used by running `semgrep --config p/trailofbits`.

## ShellCheck

ShellCheck can be installed with common system-wide package installers. For example, `brew install shellcheck` works on macOS, and `sudo apt install shellcheck` works on Ubuntu. To reproduce the results in TOB-OISY-1, run `shellcheck scripts/*.sh`.

## Vitest

Vitest is already installed by OISY's `package.json` file. However, for completeness, the executable can be installed as a development dependency with `npm install -D vitest`. The executable can thereafter be run with `npx vitest`.

## zizmor

`zizmor` can be installed using Cargo by running `cargo install zizmor --locked`. To analyze the GitHub workflows in the `oisy-wallet` repository, navigate to the root directory of the repository and run `zizmor .`. To enable pedantic output, run the tool with the `-p` flag.

# D. Non-Security-Related Recommendations

The following recommendations are not associated with any specific vulnerabilities. However, they will enhance code readability and may prevent the introduction of vulnerabilities in the future.

- **Make the instructions for running the front end more evident.** The commands in figure F.1 appear to be required. However, nowhere in the documentation do these commands appear together.

```
dfx start --background
npm ci
npm run deploy
npm run build
npm run dev
```

*Figure D.1: Commands required to run the front end*

- **Make the order of the fields in the struct patterns in figures F.2 and F.3 match their corresponding struct declarations.**

```
let StoredUserProfile {
    created_timestamp,
    updated_timestamp,
    version,
    credentials,
    settings,
} = user;
```

*Figure D.2: The order of the fields in this struct pattern does not match `StoredUserProfile`'s declaration. (`src/shared/src/impls.rs#222–228`)*

```
let CanisterStatusResponse {
    status,
    module_hash,
    settings,
    memory_size,
    cycles,
    idle_cycles_burned_per_day,
    ..
} = value;
```

*Figure D.3: The order of the fields in this struct pattern does not match `CanisterStatusResponse`'s declaration. (`src/shared/src/std_canister_status.rs#33–41`)*

- **Run `cargo clippy --all-targets -- -W clippy::pedantic` and address the warnings it produces.** Note that without `--all-targets`, Clippy's pedantic lints produce no warnings.

- **Fix the `validates_on_deserialize` test, which currently fails.** The test fails on the test vector in figure F.4. The symbol's length is 23, which is greater than `MAX_SYMBOL_LENGTH` (20).

```
24    TestVector {
25        input: SplToken {
26            token_address: SplTokenId("1".repeat(32)),
27            symbol: Some("☃️☃️☃️ ☃️ ☃️☃️☃️".to_string()),
28            decimals: Some(6),
29        },
30        valid: true,
31        description: "Valid SplToken",
32    },
```

*Figure D.4: Test vector causing the `validates_on_deserialize` test to fail (`src/shared/src/types/tests.rs#24–32`)*

- **Rewrite the code in figure F.5 as in figure F.6 or similar.** Use of the term `all_utxos_found` is misleading, because the variable is set to true when *no* UTXOs are found.

```
117    let all_utxos_found = pending_transaction
118        .utxos
119        .iter()
120        .all(|utxo| !current_utxos.contains(utxo));
121    !is_old && !all_utxos_found
```

*Figure D.5: Excerpt of the `prune_pending_transactions` function (`src/backend/src/heap_state/btc_user_pending_tx_state.rs#117–121`)*

```
let some_utxo_found = pending_transaction
    .utxos
    .iter()
    .any(|utxo| current_utxos.contains(utxo));
!is_old && some_utxo_found
```

*Figure D.6: Proposed rewrite of the code in figure F.5*

- **Fix error message on anonymous principal.** The error message should be updated to say "...is not supported."

```
if (identity.getPrincipal().isAnonymous()) {
  return { success: false,
    err: 'Using the dapp with an anonymous user if not supported.' };
}
```

*Figure D.7: `src/frontend/src/lib/services/address.services.ts#142–144`*

- **Avoid nesting ternary operators.** Nesting ternary operators makes the code harder to read and should be avoided.

```
const txBlockIndex = !isSourceTokenIcrc2
  ? standard === 'icrc'
    ? await sendIcrc({
        ...transferParams,
        ledgerCanisterId
      })
    : await sendIcp(transferParams)
    : undefined;
```

*Figure D.8:* `src/frontend/src/lib/services/swap.services.ts#60–67`

## E. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From September 3 to September 5, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the DFINITY team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the 14 issues described in this report, DFINITY has resolved nine issues, has partially resolved two issues, and has not resolved the remaining three issues. For additional information, please see the Detailed Fix Review Results below.

| ID | Title | Severity | Status |
|---|---|---|---|
| 1 | ShellCheck warnings | Informational | Resolved |
| 2 | Tests require a script to run | Informational | Unresolved |
| 3 | Tests do not work with the latest version of pocket-ic | Informational | Partially Resolved |
| 4 | Reliance on vulnerable dependencies | Informational | Resolved |
| 5 | Deployment workflow is vulnerable to cache poisoning | Informational | Unresolved |
| 6 | Potential credential persistence in artifacts | Informational | Resolved |
| 7 | Unpinned external GitHub CI/CD action versions | Low | Resolved |
| 8 | The top_up_cycles_ledger API may return invalid data | Low | Resolved |
| 9 | Codebase contains obsolete code | Informational | Resolved |
| 10 | HTML tags matched with regular expressions | Informational | Partially Resolved |

| | | | |
|---|---|---|---|
| 11 | Transaction IDs are not validated by the back end canister | Low | Resolved |
| 12 | OISY reuses the identity key from browser storage on sign in | Low | Resolved |
| 13 | Measurable test coverage is low | Undetermined | Unresolved |
| 14 | Front end may misrepresent bitcoin transactions with multiple outputs | Medium | Resolved |

## Detailed Fix Review Results

**TOB-OISY-1: ShellCheck warnings**

Resolved in commit 238d245, commit 17358d9, and PR #8091. The first commit addresses the existing `shellcheck` warnings, the second commit adds `shellcheck` to the GitHub CI pipeline, and the PR enforces shell script linting on each new pull request. Running the tool on all shell scripts under the scripts directory produces no new warnings.

**TOB-OISY-2: Tests require a script to run**

Unresolved. The team has decided not to address this issue. The reason given is that the team does not want to download a Wasm canister when `cargo test` runs, which necessitates an external setup process. However, the team is looking into alternatives to make this setup more robust.

**TOB-OISY-3: Tests do not work with the latest version of pocket-ic**

Partially resolved in PR #6484. The PR updates the version of PocketIC used to 8.0.0. However, the latest version (as of June 6, 2025) is version 9.0.3.

**TOB-OISY-4: Reliance on vulnerable dependencies**

Resolved in PR #6349. The PR updates the version of the `cargo-binstall` dependency used by the project to 0.21.2, which also updates the two transitive dependencies `idna` and `rustls` with known vulnerabilities.

**TOB-OISY-5: Deployment workflow is vulnerable to cache poisoning**

Unresolved. The DFINITY team has decided to accept this risk. Currently, every deployment to a production environment is compared to a local deployment through Docker. Since the project has reproducible builds, any discrepancies would be identified at that time.

**TOB-OISY-6: Potential credential persistence in artifacts**

Resolved in commit e47d165 and PR #5730. Commit `e47d165` adds `persist-credentials: false` to all checkout actions. PR #5730 also adds Zizmor to the GitHub CI pipeline.

**TOB-OISY-7: Unpinned external GitHub CI/CD action versions**
Resolved. All external GitHub actions are now pinned to a specific commit. PR #5730 also adds Zizmor to the GitHub CI pipeline, which ensures that unpinned actions will be flagged in the future.

**TOB-OISY-8: The top_up_cycles_ledger API may return invalid data**
Resolved in commit `0993fa5`. The return value from the `top_up_cycles_ledger` API now contains the correct values from the cycle and back end canisters.

**TOB-OISY-9: Codebase contains obsolete code**
Resolved in PR #6237. The unused migration code has been removed.

**TOB-OISY-10: HTML tags matched with regular expressions**
Partially resolved in PR #5966 and PR #6030. The two PRs attempt to make HTML parsing more robust by making the regular expression insensitive to case and trailing whitespaces. However, parsing HTML using regular expressions is still error-prone, and we still recommend that DFINITY replace this approach with a dedicated parser.

**TOB-OISY-11: Transaction IDs are not validated by the back end canister**
Resolved in PR #6423, PR #6432, and PR #6483. The first PR adds validation of BTC transaction IDs, UTXOs, and addresses when the corresponding request is deserialized. The second and third PRs add similar validation for user profiles, user credentials, and user-defined tokens.

**TOB-OISY-12: OISY reuses the identity key from browser storage on sign in**
Resolved in PR #8398. The front end now deletes any identity keys found in storage before a new `AuthClient` is created.

**TOB-OISY-13: Measurable test coverage is low**
Unresolved. Most of the implemented tests still rely on the PocketIC test runner. However, as mentioned in the issue description, there is no way to measure test coverage for such tests. This means that measurable test coverage remains low.

**TOB-OISY-14: Front end may misrepresent bitcoin transactions with multiple outputs**
Resolved in PR #5627. The front end now lists all recipient addresses and correctly computes the total output amount when displaying Bitcoin transactions.

## F. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

| Fix Status | |
|---|---|
| **Status** | **Description** |
| Undetermined | The status of the issue was not determined during this engagement. |
| Unresolved | The issue persists and has not been resolved. |
| Partially Resolved | The issue persists but has been partially resolved. |
| Resolved | The issue has been sufficiently resolved. |

## About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries and government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on X and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact or email us at info@trailofbits.com.

**Trail of Bits, Inc.**
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

# Notices and Remarks

## Copyright and Distribution

## Test Coverage Disclaimer

Trail of Bits performed all activities associated with this project in accordance with a statement of work and an agreed-upon project plan.

Security assessment projects are time-boxed and often rely on information provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test software controls and security properties. These techniques augment our manual security review work, but each has its limitations. For example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. A project's time and resource constraints also limit their use.