Stardust

Security Audit

July 30th 2021

Version 1.0.0

Prepared by

Optilistic

Optilistic

Introduction

This document includes the results of the preliminary security audit for Urbit's smart contract code as found in the section titled 'Source Code'. The security audit was performed by the Optilistic team from July 19th 2021 to July 30th 2021.

The purpose of this audit is to review the source code of the Stardust Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Optilistic's review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

We've identified a few issues of medium and low severity that should be considered to improve the longevity of the project.

We did not identify any issues of high severity that would substantially compromise the integrity of the project.

Specification

Our understanding of the specification was based on the following sources:

- Stardust Project Scope
- Direct discussions with the Urbit team
- Comments in the contract source code / README
- Various urbit blog posts

Source Code

The following source code was reviewed during the audit:

Repository	Commit
Github	c446b1f12f53fa75ea6c347daee1e15df562a81d

Specifically, we audited the following contracts:

Contract	Sha256
IAzimuth.sol	7ef2451f17083000aa7fec31ea0135c2cfb2c245a1fe078bbc389e5fecf21311
IEcliptic.sol	12ed5311d0df888bc94b417c36e4a9b3a9eb1be4c8b03fb23f35f321e558d446
StarToken.sol	693b7937db9099d38e14d11efec2f41bde50afe9dd268513cc19b5424f2ad12a
Treasury.sol	c83cc46f5d0c34509a3400fecfcc43ae838e891107b86af3cf4ff25a8f4c2506

Note: This document contains an audit solely of the Solidity contracts listed above, including the reading of relevant code in Azimuth and Ecliptic smart contracts. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Methodology

The audit was conducted in several steps.

First, we reviewed in detail all available documentation and specifications for the project, as described in the 'Specification' section above.

Second, we performed a thorough manual review of the code, checking that the code matched up with the specification, as well as the spirit of the contract (i.e. the intended behavior). During this manual review portion of the audit we primarily searched for security vulnerabilities, unwanted behavior vulnerabilities, and problems with systems of incentives.

Third, we performed the automated portion of the review consisting of measuring test coverage (while also assessing the quality of the test suite) and evaluating the results of various symbolic execution tools against the code.

Lastly, we performed a final line-by-line inspection of the code – including comments –in effort to find any minor issues with code quality, documentation, or best practices.

Issues Descriptions and Recommendations

Severity Level Reference

Level	Description
High	The issue poses existential risk to the project, and the issue identified could lead to massive financial or reputational repercussions.
Medium	The potential risk is large, but there is some ambiguity surrounding whether or not the issue would practically manifest.
Low	The risk is small, unlikely, or not relevant to the project in a meaningful way.
Code Quality	The issue identified does not pose any obvious risk, but fixing it would improve overall code quality, conform to recommended best practices, and perhaps lead to fewer development issues in the future.

[STR-01] Galaxy owners can "farm" star sponsorships

MEDIUM

The **Treasury.sol** contract's **deposit** function is not concerned with escapes. Consequently, a galaxy owner can "farm" star sponsorships by performing the following:

1. Deposit a star
2. Wait for another star deposit
3. Redeeming the newer star
4. Escape the star to itself
5. Re-deposit the star
6. Go to step 2 of this list

This does not affect the integrity of Stardust itself, but it could incentivize toxic behavior now or in the future if/when there are incentives for galaxies to increase the number of sponsorships they have.

Consider: Update Treasury.sol's **deposit** function to only accept stars that have not been escaped.

[STR-02] Heightened chance of lost stars

LOW

StarToken.sol is an ERC-777, which allows users to trade STAR in small increments. Because of this, there is generally a higher chance of stars getting "stuck" in **Treasury.sol** when accounts lose access to their STAR.

We consider this to be of low severity since the same reasoning can be applied to accounts that hold stars directly. However, at least one star is highly likely to be stuck in Treasury.sol; any single fraction of STAR lost – by any account – would immediately cause the first star in the Treasury.sol to be permanently unredeemable.

Consider: Update StarToken.sol's **constructor** to mint some amount of STAR to the current Ecliptic contract, in case Urbit's senate wants to deal with stuck stars in the future.

[STR-03] Redundant authorization logic

LOW

The **Treasury.sol** contract's **redeem** function has the following check:

```
require(azimuth.isOwner(_star, address(this)));
```

This check is also done in **Azimuth.canTransfer**.

Consider: Remove this check and leave authorization entirely to **Ecliptic.transferPoint**, in the rare chance Ecliptic's authorization logic diverges from this check in the future (due to a contract upgrade), which would cause Treasury to unnecessarily restrict its outbound transfers.

Removing this line would also increase branch coverage, as it's currently not possible to write a test case to make this require statement fail.

[STR-04] Function visibility

CODE QUALITY

Consider changing the visibility of Treasury.sol's **deposit** and **redeem** functions from **public** to **external**, communicating and enforcing that this contract will not be depositing any stars to itself, as well as saving gas for senders.

Consider making the same change to StarToken.sol's **mint** and **ownerBurn** functions to also save on gas.

[STR-05] Bit size discrepancy

CODE QUALITY

The **Treasury.sol** contract deals with stars in **uint16** instead of Ecliptic's **uint32** for general points. If this is guaranteed to be safe, consider writing a comment explaining why.

Upgradability Options (added Aug 3rd)

If upgradability for Stardust is desired, there are three main options:

Full Upgradability

Just like the relationship between Azimuth, a data layer contract, and Ecliptic, a business logic layer contract, you could split Treasury.sol into these two separate contract types.

- **Pro:** You gain full upgradability, allowing you to upgrade the logic behind Treasury to anything you need.
- **Con:** Complexity is greatly increased. In addition to the split, you would also need to add a community-trusted way of upgrading the contract, e.g. governance.

Delegated Upgradability

Similar to **full** upgradability, this solution would split Treasury.sol into two separate contract types. However, instead of implementing *separate* governance, you can *delegate* governance to Ecliptic instead, allowing upgrades to Ecliptic to also include upgrades to Stardust.

- **Pro:** Less complex than full upgradability (no governance coding).
- **Con:** Still a significant enough change to require another round of audits.

Delegated Ownership

Rather than implementing upgradability, a perhaps better option would be to update Treasury.sol to inherit **Ownable,** set Ecliptic as the owner, and allow the owner to manipulate **assets** directly.

- **Pro:** Far less complexity than implementing upgradability, and would not require another round of audits.
- **Pro:** Allows the Urbit Senate to upgrade Ecliptic to deal with Stardust's assets in the future. i.e. solving [STR-02]
- **Con:** Cannot upgrade the business logic of Treasury.sol.

Test Coverage

Steps Taken

- `$ npm install --save-dev solidity-coverage`
- Added `require('solidity-coverage')` to **hardhat.config.js**

Test Results

Output of running `npm test` at the root of the project directory:

```
Compiling 20 files with 0.4.24
azimuth-solidity/contracts/EclipticBase.sol:56:3: Warning:
Function state mutability can be restricted to view
  function onUpgrade()
  ^ (Relevant source part starts here and spans across multiple
lines).

Compiling 14 files with 0.8.4
Compilation finished successfully



  StarToken
    ✓ has a name
    ✓ has a symbol
    ✓ assigns the initial total supply to the creator
    ✓ should contain zero balance once deployed (44ms)
    ✓ allows operator burn (48ms)

  Treasury
0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
0x70997970C51812dc3A010C7d01b50e0d17dc79C8
0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC
```

```
      ✓ has no assets when deployed
      ✓ knows how to find azimuth
      ✓ deploys a token contract with no initial supply
      ✓ allows deposit of star from owner (176ms)
      ✓ allows retrieval of all assets in order
      ✓ doesn't allow deposit of non-stars
      ✓ doesn't allow deposit from non-owner (53ms)
      ✓ allows redeem from token holder (162ms)
      ✓ allows deposit-send-redeem pattern (207ms)
      ✓ doesn't allow redeem from non-holder
      ✓ doesn't allow redeem when balance is too low (129ms)
      ✓ doesn't allow inbound safe transfer (164ms)


  17 passing (3s)
```

Code Coverage

Code coverage was measured by running **solidity-coverage** at the root of the project.

File	%Stmts	%Branch	%Funcs	%Lines	Uncovered Lines
IAzimuth.sol	100	100	100	100	
IEcliptic.sol	100	100	100	100	
StarToken.sol	100	100	100	100	
Treasury.sol	100	83.33	100	100	
All Files	100	83.33	100	100	

Evaluation

The code coverage for the Urbit repository is good. Resolving STR-03 will bring it even closer to 100%.

Automated Analysis

Mythril

Mythril is a security analysis tool that uses concolic analysis, taint analysis, and control flow checking to detect a variety of security vulnerabilities.

In order to run Mythril against the codebase we performed the following steps:

-
  ```
  $ docker run -v $(pwd):/tmp mythril/myth analyze /tmp/contracts/StarToken.sol
  ```

-
  ```
  $ docker run -v $(pwd):/tmp mythril/myth analyze /tmp/contracts/Treasury.sol
  ```

Mythril reported a successful analysis and had no issues to report. For the record, this does not mean there are zero issues with the code, only that this tool did not find any.

Slither

Slither is a solidity static analysis framework. It detects many vulnerabilities, from high threats to benign ones, of which there are usually many.

In order to run Slither against the codebase we performed the following steps:

-
  ```
  $ slither . --filter-paths "Address.sol|AddressUtils.sol|Azimuth.sol|Claims.sol|Ecliptic.sol|EclipticBase.sol|ERC721.sol|ERC721Basic.sol|ERC721Receiver.sol|ERC777.sol|Ownable.sol|Polls.sol|SafeMath.sol"
  ```

Slither identified some benign reentrancy vulnerabilities, but manual inspection revealed them to be false positives.

Slither also identified some code quality issues, which we recorded in the "Issues Descriptions and Recommendations" section.

Appendix

Exhibit A - Dependency Graphs

Black, filled-in circles represent “inherits from” relationships and white circles represent “imports” relationships.

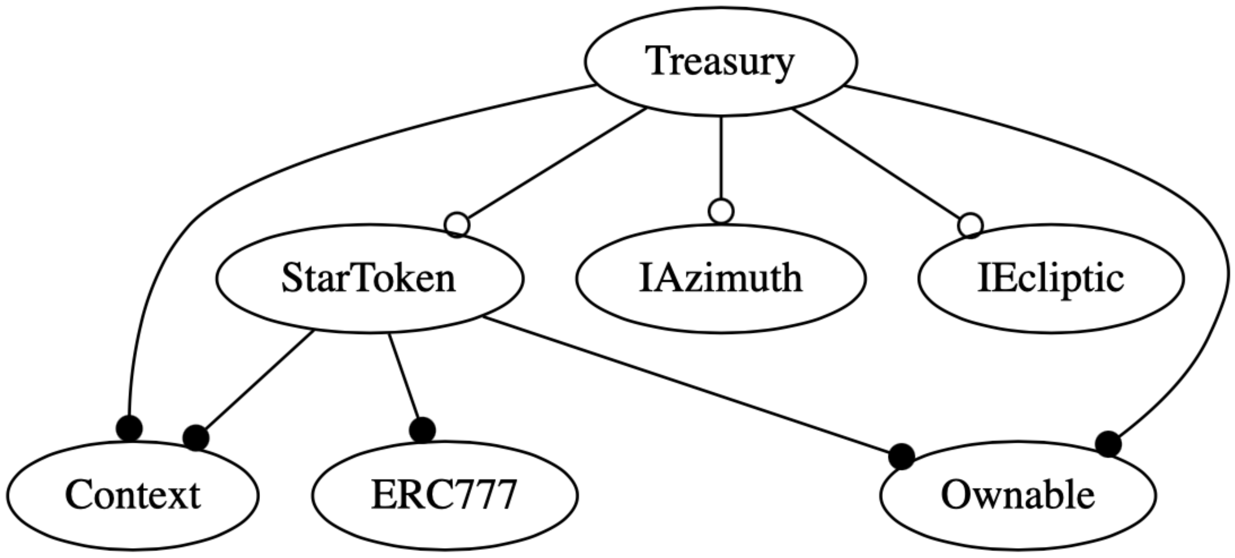

Exhibit B - Suggested Renamings, Refactors and Cleanups

Treasury

- The constant **oneStar** in **Treasury.sol** should be cased as **ONE_STAR** by convention.

StarToken

- No issues found.

Exhibit C - Comment and Documentation Typos and Potential Improvements

Treasury

- Line 12: Comment appears to be incomplete.
- Line 112: Comment reads “star tokens” (plural), which is inconsistent with other comments that read in the singular.
- Line 125: Consider adding spaces around the ‘greater than’ operator to be stylistically consistent with line 122.

StarToken

- No issues found.

Exhibit D -Disclaimer

Optilistic makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Optilistic specifically disclaims all implied warranties or merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Optilistic will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand against company by any other party. In no event will Optilistic be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Optilistic has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Urbit team and only the source code Optilistic notes as being within the scope of Optilistic's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Optilistic. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Optilistic is not responsible for the content or operation of such websites, and that Optilistic shall have no liability to your or any other person or entity for the use of third party websites. Optilistic assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.