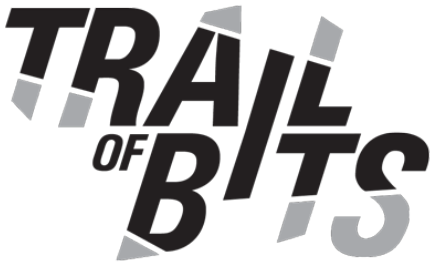

Liquity Proxy

Rapid Code Review

March 5, 2021

Prepared For:
Robert Lauko | *Liquity*
robert@liquity.org

Rick Pardoe | *Liquity*
rick@liquity.org

Prepared By:
Natalie Chin | *Trail of Bits*
natalie.chin@trailofbits.com

Review Summary

From February 22, 2021 to February 26, 2020, Trail of Bits performed an assessment of the Liquity Proxy implementation with one engineer, targeting commit b90440d. We reported eight issues ranging from high to undetermined severity.

We evaluated the Liquity proxy for flaws and attack vectors such as:

- Can an attacker easily phish users to execute arbitrary code on their Proxy to steal funds?
- Can the user's proxy be self-destructed?
- Can an attacker abuse deviations from Maker's mainnet proxy and Liquity's Proxy?

Of the findings reported, two high-severity issues pertained to the use of `delegatecall` in the DSProxy; these could result in the proxy being self-destructed and losing funds. Two low-severity issues involve dependencies that were copy-pasted in their entirety rather than imported externally, which makes it difficult to keep them in sync with upstream patches which might otherwise fix known vulnerabilities. Another low-severity issue pertained to lack of data validation on arguments provided by external users, which may result in a loss of user funds. An informational issue was raised regarding the lack of events for functions in the script, which would be beneficial to have for monitoring purposes as these scripts increase in complexity. Lastly, an issue of undetermined severity pertained to these scripts being publicly visible, which could make it easier for attackers to write exploits to abuse the Liquity Protocol.

On the following page, we review the maturity of the codebase and the likelihood of future issues. In each area of control, we rate the maturity from strong to weak, or missing, and give a brief explanation of our reasoning. Liquity should consider the following steps to improve their security maturity:

- Use Slither to automatically detect issues in code
- Integrate `npm audit` or `yarn audit` into Liquity's Continuous Integration pipeline to ensure vulnerable dependencies are detected and remediated before each new build.
- Document a list of risks associated with users interacting with arbitrary scripts to ensure that users are aware of the risks associated with arbitrary contracts.
- Review all deviations from the proxy versions in the Liquity codebase with the Maker-deployed version on mainnet and document all situations where a user's transaction may revert.

Code Maturity Evaluation

Category Name	Description
Access Controls	**Satisfactory.** Appropriate access controls were in place in the DSProxy to execute the scripting contracts. Supplementary documentation on risks associated with interacting with these scripts would be valuable.
Arithmetic	**Satisfactory.** The contracts included the use of safe arithmetics. One minor code-quality issue was raised to revert when the LUSD amount is 0.
Assembly/Low Level	**Weak.** The contracts contained no assembly usage, but a lack of contract existence check on the proxy and opportunity for arbitrary script execution that could result in a loss of user funds.
Centralization	**Satisfactory.** There are no privileged accounts in the proxy scripts. However, additional documentation related to deployment risks and how users check deployed addresses would be advantageous.
Contract Upgradeability	**Not Applicable.** The contracts contained no upgradeability mechanisms.
Function Composition	**Strong.** Functions and contracts were organized and scoped appropriately.
Front-Running	**Strong.** No front-running issues were identified.
Monitoring	**Moderate.** The Liquity Protocol contained sufficient events, but the scripting contracts could benefit from the addition of events for more complex interactions.
Specification	**Satisfactory.** The system had comprehensive system documentation for the Liquity Protocol. Additional documentation regarding user experience and expectations would be beneficial.
Testing & Verification	**Satisfactory.** The contracts included unit tests on the proxy, which verified balances pre and post-transfer. However, `claimCollateralAndOpenTrove` was missing a test that only a proxy owner can call it to execute code.

Appendix A. Code Maturity Classifications

Code Maturity Classes	
Category Name	**Description**
Access Controls	Related to the authentication and authorization of components.
Arithmetic	Related to the proper use of mathematical operations and semantics.
Assembly Use	Related to the use of inline assembly.
Centralization	Related to the existence of a single point of failure.
Upgradeability	Related to contract upgradeability.
Function Composition	Related to separation of the logic into functions with clear purpose.
Front-Running	Related to resilience against front-running.
Key Management	Related to the existence of proper procedures for key generation, distribution, and access.
Monitoring	Related to use of events and monitoring procedures.
Specification	Related to the expected codebase documentation.
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.).

Rating Criteria	
Rating	**Description**
Strong	The component was reviewed and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.

Missing	The component was missing.
Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.