#HashEx
BLOCKCHAIN SECURITY

Fluidity Money

smart contracts final audit report

February 2022


hashex.org


contact@hashex.org

Contents

1. Disclaimer 3

2. Overview 4

3. Found issues 5

4. Contracts 6

5. Conclusion 8

Appendix A. Issues severity classification 9

Appendix B. List of examined issue types 10

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the **Fluidity Money** team to perform an audit of their smart contract. The audit was conducted between **02/02/2022** and **07/02/2022**.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at https://github.com/fluidity-money/fluidity-solana.

2.1 Summary

Project name	Fluidity Money
URL	https://www.fluidity.money/
Platform	Solana
Language	Rust

2.2 Contracts

Name	Address
https://github.com/fluidity-money/fluidity-solana	

3. Found issues

High	1 (33%)
Low	1 (33%)
Info	1 (34%)

Cdb. https://github.com/fluidity-money/fluidity-solana

ID	Severity	Title	Status
CdbIc1	High	Lack of program pubkey validation	Acknowledged
CdbIc2	Low	Lack of checks in LogTVL function	Acknowledged
CdbIc3	Info	Implicit ownership checks	Acknowledged

4. Contracts

Cdb. https://github.com/fluidity-money/fluidity-solana

Issues

CdbIc1 Lack of program pubkey validation

High Acknowledged

A user provides the program address to be called. The `wrap()` function calls passed function to transfer assets to the account, but an attacker can create and pass a malicious program that won't transfer assets and just return success. The wrap() function in such case will mint tokens without assets being transferred to the account.

Recommendations

Add missing checks for the program ownership (see arbitrary signed program invocation type of bugs).

CdbIc2 Lack of checks in LogTVL function

Low Acknowledged

The function `LogTVL()` checks the value of the value of the specified reserve and writes it to an account specified by the caller. The function has no restrictions on what reserve is passed by the caller and therefore an attacker may pass his reserve and override specified `data_account`.

CdbIc3 Implicit ownership checks

Info Acknowledged

The `wrap()` instruction ensures that the caller transfers assets to reserve and expected amount of Fluid tokens are minted. This function has no explicit check that the reserve account is actually Fluidity's reserve (this check is done implicitly the in `DepositObligationCollateral` call).

Recommendation

We recommend adding explicit check that the reserve account is actually a Fluidity's account. This will make refactoring less error prone.

5. Conclusion

1 high, 1 low and 1 informational severity issues were found.

Audit includes recommendations on improving the code and preventing potential attacks.

Appendix A. Issues severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Info.** Issues that do not impact the contract operation. Usually, info severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- Unchecked math
- Implicit visibility levels
- Weak sources of randomness
- Usage of deprecated code

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY