Fluidity

Audit Report

contact@movebit.xyz

https://twitter.com/movebit_

Mon Feb 05 2024

Fluidity Audit Report

1 Executive Summary

1.1 Project Information

Description	A yield generating system
Type	DeFi
Auditors	MoveBit
Timeline	Mon Jan 29 2024 - Mon Feb 05 2024
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/gmxbt/fluidity-sui
Commits	69caedc47c509f3ec7293a935eed97e2ad1deb17

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
MOV17	Move.toml	e1bd8e8bcfecc0b041ada69593f0217300c573b4
FCO1	sources/fluidity_coin.move	93804765eddc5c49b33bf8078710ba870930b578

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	4	3	1
Informational	0	0	0
Minor	3	3	0
Medium	1	0	1
Major	0	0	0
Critical	0	0	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Fluidity to identify any potential issues and vulnerabilities in the source code of the Fluidity smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 4 issues of varying severity, listed below.

ID	Title	Severity	Status
FCO-1	Module Lacks Interface For Upgrades	Medium	Acknowledged
FCO-2	Lack of Events Emit	Minor	Fixed
FCO-3	Code Optimization	Minor	Fixed
FCO-4	Don't Transfer Zero Coin	Minor	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the Fluidity Smart Contract :

Admin

- The `admin` can add a new Vault to the Vault through the `init_vault()` function.
- The `admin` can mint new coins to the coin reserve using the `mint()` function.
- The `admin` has the privilege to burn coins from the coin reserve via the `burn()` function.
- The `admin` can collect the yield using the `collect_yield()` function.

User

- `Users` can use the `wrap()` functions to pledge underlying assets in exchange for `fCoin` .
- `Users` can call the `unwrap()` function to exchange `fCoin` for the corresponding underlying assets.

4 Findings

FCO-1 Module Lacks Interface For Upgrades

Severity: Medium

Status: Acknowledged

Code Location:

sources/fluidity_coin.move#150-195

Descriptions:

The module uses a third-party contractual interface and lacks an interface for upgrading, which may result in the module not being able to be modified when the third-party interface function changes.

Suggestion:

In order to take into consideration future module changes, it is recommended to add an interface for contract upgrades.

FCO-2 Lack of Events Emit

Severity: Minor

Status: Fixed

Code Location:

sources/fluidity_coin.move#124-150   258

Descriptions:

The smart contract lacks appropriate events for monitoring sensitive operations, which could make it difficult to track sensitive actions or detect potential issues.

Suggestion:

It is recommended to emit events for those sensitive functions.

FCO-3 Code Optimization

Severity: Minor

Status: Fixed

Code Location:

sources/fluidity_coin.move#104,131,144,160-161,206-207

Descriptions:

There are statements in the contract that use Boolean variables to compare with Boolean values, such as `global.paused == false`, and it is recommended that just use the value of that field directly.

Suggestion:

It is recommended to fix them and increase the readability of the code.

FCO-4 Don't Transfer Zero Coin

Severity: Minor

Status: Fixed

Code Location:

sources/fluidity_coin.move#237

Descriptions:

After calling `redeem`, if `redeemed_coin_amount` is 0, the protocol will transfer zero coin to the user, which will lead to more useless objects under the address, it is recommended to call `destroy_zero` in the contract to destroy coin with a zero value.

Suggestion:

It is recommended to call `destroy_zero` in the contract to destroy coin with a zero value.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.