Security Assessment

Flux III

Apr 7th, 2021

Summary

This report has been prepared for Flux III Final smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Additionally, this audit is based on a premise that all the external contracts are implemented safely.

The security assessment resulted in 5 findings that ranged from minor to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Flux III
Description	Defi;Stake
Platform	Conflux
Language	Solidity
Codebase	https://github.com/01-finance/flux-protocol
Commits	https://github.com/01-finance/flux-protocol/commit/539ee39c3e89a60b6603aa0d0f6742ebcc6804c5

Audit Summary

Delivery Date	Apr 07, 2021
Audit Methodology	Manual Review
Key Components	

Vulnerability Summary

Total Issues	5
Critical	0
Major	0
Minor	2
Informational	3
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
LPT	LPTokenProxy.sol	90a0f2befd29903a094b533ba926bbc1196050af5487af638fbbf3d1a2e00d6d
STA	Stake.sol	349b2448fdd1fb6db9eda4dcbf23859a5b1761c6b737e6416271f1d7cea1a7f3

Centralization

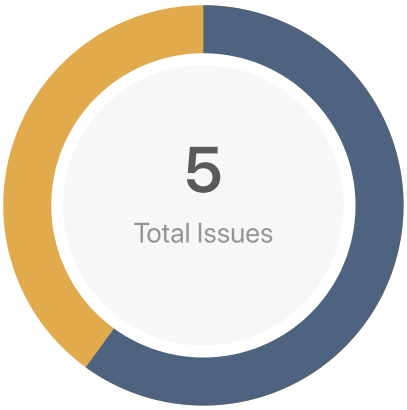
[Stake.sol]

- owner can redeem all staked tokens to `Ethereum`.
- owner can supply assets to `Flux`.

[LPTokenProxy.sol]

- owner can withdraw assets from `Compound` and transfer them to `Flux`.
- owner can withdraw assets from `Aave` and transfer them to `Flux`.

Findings



Critical	0 (0.00%)
Major	0 (0.00%)
Minor	2 (40.00%)
Informational	3 (60.00%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
STA-1	Shadowed Declaration	Coding Style	Informational	Resolved
STA-2	Unused Safe Check	Logical Issue	Informational	Resolved
STA-3	Missing Some Important Checks	Logical Issue	Minor	Resolved
STA-4	Redundant State Variable	Logical Issue	Informational	Resolved
STA-5	Inaccurate Value of totalStake	Logical Issue	Minor	Resolved

STA-1 | Shadowed Declaration

Category	Severity	Location	Status
Coding Style	Informational	Stake.sol: 102, 114, 123, 132	Resolved

Description

Later declaration shadows a previous declaration. For example, `last` was already declared in L102. But in L114, a new variable with the same name is declared.

```
function stakeAmountAt(address staker, uint256 snapshotId) public view returns (uint256
amount) {
        uint256 last = lastSnapshotId;
        ......
        uint256 last = snapshotId == 1 ? 0 : balanceOfAt(staker, snapshotId - 1);
        ......
}
```

Recommendation

Consider using a different name to avoid shadowing.

Alleviation

The team had changed logic of functions `stakeAmountAt()` and `totalStakeAt()`, so this issue isn't exist. Code change was applied in commit 4cab0d171612f56508c9c26ea95f9734d2fe5847.

STA-2 | Unused Safe Check

Category	Severity	Location	Status
Logical Issue	Informational	Stake.sol: 80~81	Resolved

Description

The return value of `totalSupply()` wasn't used by any check.

```
//safe check
_underlyingToken0.totalSupply();
_underlyingToken1.totalSupply();
```

Recommendation

Consider removing them or adding logic to check their values.

Alleviation

The team had removed these unused safe check code. Code change was applied in commit 4cab0d171612f56508c9c26ea95f9734d2fe5847.

STA-3 | Missing Some Important Checks

Category	Severity	Location	Status
Logical Issue	Minor	Stake.sol: 124, 130	Resolved

Description

If `lastSnapshotId` == 0, `snapID` should be 1 or `uint(-1)`. Besides, `snapID` cannot be larger than `lastSnapshotId` + 1.

Recommendation

Consider adding related check like what `stakeAmountAt()` did.

Alleviation

The team had change the logic of function `totalStakeAt()`, so this issue isn't exist. Code change was applied in commit 4cab0d171612f56508c9c26ea95f9734d2fe5847.

STA-4 | Redundant State Variable

Category	Severity	Location	Status
Logical Issue	Informational	Stake.sol: 147, 162, 257	Resolved

Description

`manager` was declared and assgined value `fluxApp` several times in different functions. Actually, these duplicate codes are unnecessary. Using `fluxApp` to implements related functions is enough.

```
IStakePoolManager manager = fluxApp;
```

Recommendation

Consider removing `manager` and using `fluxApp` instead of it.

Alleviation

The team heeded our advice and changed related code. Code change was applied in commit 4cab0d171612f56508c9c26ea95f9734d2fe5847.

STA-5 | Inaccurate Value of totalStake

Category	Severity	Location	Status
Logical Issue	Minor	Stake.sol: 152	Resolved

Description

When user stake token, related `amount` was added to the `totalStake`. But when user unstake token, related `amount` wasn't deducted from the `totalStake`.

Recommendation

Consider deducting related `amount` from the `totalStake` when user unstake token.

Alleviation

The team heeded our advice and changed related code. Code change was applied in commit 4cab0d171612f56508c9c26ea95f9734d2fe5847.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.