Go Cash

Security Assessment

March 18th, 2021

For :
Go Cash

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are 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 review.

CertiK Reports do 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.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing 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.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

Overview

Project Summary

Project Name	Go Cash
Description	Stable coin staking
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commit	ac1383d917562411d2dcd50663ee45de507e40c2

Audit Summary

Delivery Date	Mar. 18th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Mar. 12th, 2021 - Mar. 15th, 2021

Vulnerability Summary

Total Issues	7
Total Critical	0
Total Major	0
Total Minor	2
Total Informational	5

Executive Summary

This report has been prepared for **Go Cash** smart contract 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.

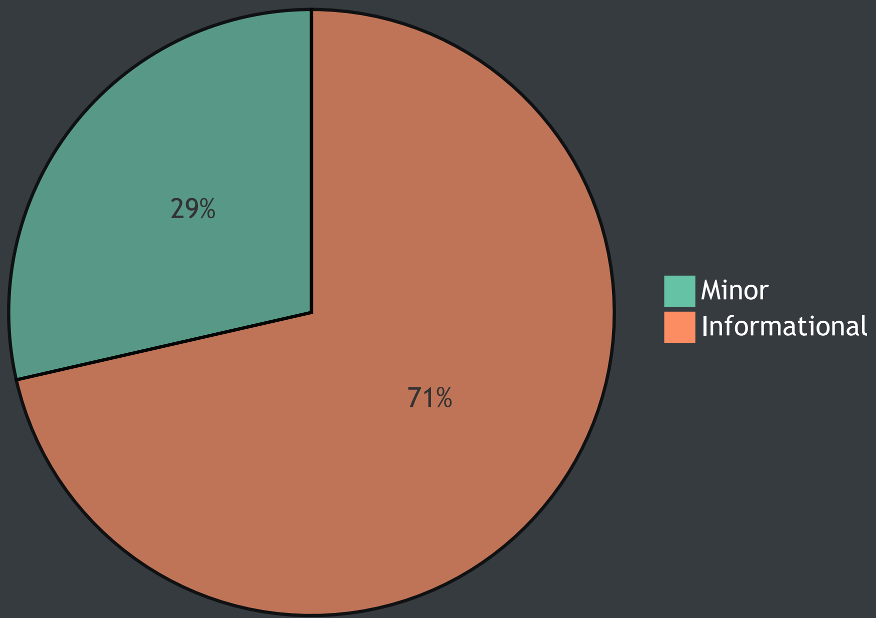
The auditing process pays special attention to the following considerations:

- 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.

File in Scope

ID	Contract	Contract SHA-256 Checksum
LTP	LPTokenPool.sol	a9c9fc510bf948aa2774752502997c0281c67ce6d11ecaa2b05587c4a666884c

Findings



ID	Title	Type	Severity	Resolved
LTP-01	Unlocked Compiler Version Declaration	Language Specific	Informational	!
LTP-02	Variable That Could Be Declared `constant`	Gas Optimization	Informational	!
LTP-03	Incorrect Naming Convention	Coding Style	Informational	!
LTP-04	Unnecessary Assignment	Optimization	Informational	!
LTP-05	Greater-Than Comparison with Zero	Optimization	Informational	!
LTP-06	Missing Zero Address Checks	Logical Issue	Minor	!
LTP-07	Reward May Not Be Available	Logical Issue	Minor	!

LTP-01: Unlocked Compiler Version Declaration

Type	Severity	Location
Language Specific	Infomational	LPTokenPool.sol L2

Description:

The compiler version utilized throughout the project uses the “^” prefix specifier, denoting that a compiler at or above the version included after the specifier should be used to compile the contracts. The compiler version should be consistent throughout the codebase.

Recommendation:

We recommend locking the compiler the lowest possible version that supports all the capabilities wished by the codebase.

Alleviation:

No alleviation.

LTP-02: Variable That Could Be Declared `constant`

Type	Severity	Location
Optimization	Infomational	LPTokenPool.sol L78

Description:

The variable is never altered beyond declaration.

Recommendation:

We recommend declaring DURATION as `constant` for gas optimization.

Alleviation:

No alleviation.

LTP-03: Incorrect Naming Convention

Type	Severity	Location
Optimization	Infomational	LPTokenPool.sol L80, L82

Description:

Solidity defines a naming convention that should be followed. In general, the following naming conventions should be utilized in a Solidity file:

- Functions and parameters should be in mixedCase

In case the naming conventions are not followed, there should be proper documentation to explain the naming and the purpose of the variable.

For example: `starttime`

Recommendation:

We recommend using `startTime` instead of `starttime` .

The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

Alleviation:

No alleviation.

LTP-04: Unnecessary Assignment

Type	Severity	Location
Optimization	Informational	LPTokenPool.sol L82, L84

Description:

Declared variables are assigned zero by default in solidity and there is no need to write explicitly.
For examples:
`uint256 public periodFinish = 0;`

`uint256 public rewardRate = 0;`

Recommendation:

We recommend declaring the variables without explicit assignment.

```
    uint256 public periodFinish;
    uint256 public rewardRate;
```

Alleviation:

No alleviation.

LTP-05: Greater-Than Comparison with Zero

Type	Severity	Location
Optimization	Informational	LPTokenPool.sol L197, L215, 239

Description:

When comparing variables of unsigned type, it's more efficient gas-wise, while taking into account that any value other than zero is indeed valid.

Recommendation:

We recommend changing the condition to check inequality with zero, as it is more efficient regarding unsigned

```
require(amount != 0, 'HUSDGOCLPTokenSharePool: Cannot stake 0');
```

Alleviation:

No alleviation.

LTP-06: Missing Zero Address Checks

Type	Severity	Location
Logical Issue	Minor	LPTokenPool.sol L105

Description:

The constructor uses addresses without validation. Discretion is advised when dealing with them.

```
constructor(
        address token_,
        address lptoken_,
        uint256 starttime_
    ) public {
        token = IERC20(token_);
        lpt = IERC20(lptoken_);
        starttime = starttime_;
    }
```

Recommendation:

We recommend using `require` statements to make sure the address arguments are not zero addresses.

```
constructor(
        address token_,
        address lptoken_,
        uint256 starttime_
    ) public {
        require(token_ != address(0), "Invalid Token Address!");
        require(lptoken_ != address(0), "Invalid LP Token Address!");
        ......
    }
```

Alleviation:

No alleviation.

LTP-07: Reward May Not Be Available

Type	Severity	Location
Logical Issue	Minor	LPTokenPool.sol

Description:

The reward distributor may proclaim a reward amount that exceeds `token` balance through `notifyRewardAmount()` and stakeholders will not be able to get paid from `token` .

Recommendation:

We would like to inquire about further precautions against such potential inconsistency between promised reward and token balance.
For example, we recommend minting enough tokens at the same time as reward being notified:

```
function notifyRewardAmount(uint256 reward)
        external
        override
        onlyRewardDistribution
        updateReward(address(0))
    {

        token.mint(address(this), reward);
        ......
    }
```

Alleviation:

No alleviation.

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 instorage 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.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

 : Issue resolved

 : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

 : Issue partially resolved. Not all instances of an issue was resolved.