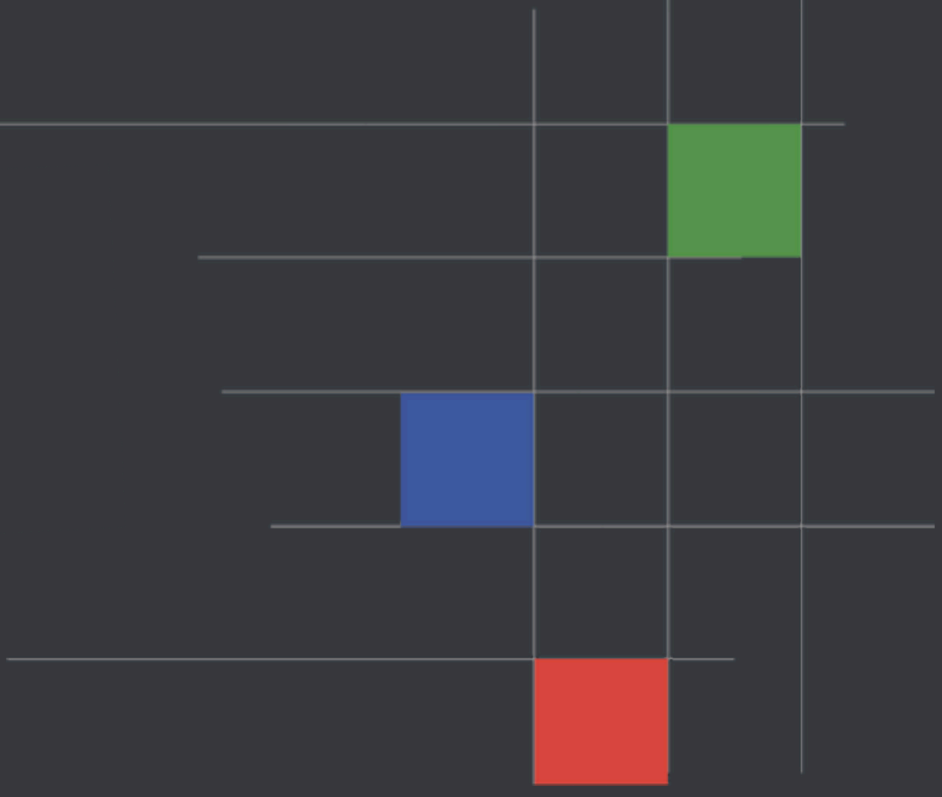
Reflect Finance

Security Assessment

January 18th, 2021



For :
Reflect Finance

By :

Dan She @ CertiK
dan.she@certik.org

Connie Lam @ CertiK
connie.lam@certik.org

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	Reflect
Description	A token contract
Platform	Ethereum
Codebase	GitHub Repository
Commit	47e0749a06e70d6538293855f950700e23d5138c

Audit Summary

Delivery Date	Jan. 18th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Jan. 10, 2021 - Jan. 18, 2021

Vulnerability Summary

Total Issues	4
Total Critical	0
Total Major	0
Total Minor	1
Total Informational	3

Executive Summary

This report has been prepared for **Reflect** 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

Contract	SHA-256 Checksum	Commit
REFLECT.sol	5481363292067ac9007be674c86aaf13c250a911999dcc4ff4f8f1461494429d	47e0749a06e70d6538293855f950700e23d5138c

---

Documentation

The sources of truth regarding the operation of the contracts in scope were lackluster and are something we advise to be enriched to aid in the legibility of the codebase as well as project. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the **CascadeV2** team or reported an issue.

Review Notes

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies, however 2 major and 2 minor vulnerabilities were identified during our audit that solely concerns the specification.

Certain discrepancies between the expected specification and the implementation of it were identified and were relayed to the team, however they pose no type of vulnerability and concern an optional code path that was unaccounted for.

The project has adequate doumentation and specification outside of the source files, also the code comment coverage is detailed.

Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, enforce linters and / or coding styles as well as correct any spelling errors and mistakes that appear throughout the code to achieve a high standard of code quality and security.

Findings

ID	Title	Type	Severity	Resolved
Exhibit-01	Function and variable types	Gas Optimization	Informational	
Exhibit-02	Incorrect error message	Optimization	Minor	
Exhibit-03	Redundant code	Optimization	Informational	
Exhibit-04	Dynamic rate between `rSupply` and `tSupply`	Logical Issue	Informational	

Exhibit-01: Function and variable types

Type	Severity	Location
Gas Optimization	Informational	

Recommendation:

The following variables should be constant:

- `_decimals`
- `_name`
- `_symbol`

The following functions should be declared `external` for lower gas costs if they will not be called internally:

- `name()`
- `symbol()`
- `decimals()`
- `totalSupply()`
- `balanceOf(address)`
- `transfer(address,uint256)`
- `allowance(address,address)`
- `approve(address,uint256)`
- `transferFrom(address,address,uint256)`
- `increaseAllowance(address,uint256)`
- `decreaseAllowance(address,uint256)`
- `isExcluded(address)`
- `totalFees()`
- `reflect(uint256)`
- `reflectionFromToken(uint256,bool)`

Exhibit-02: Incorrect Error Message

Type	Severity	Location
Optimization	Minor	REFLECT.sol L133

Description:

The error message in `require(_isExcluded[account], "Account is already excluded")` does not describe the error correctly.

Recommendation:

The message `"Account is already excluded"` can be changed to `"Account is not excluded"` .

Exhibit-03: Redundant Code

Type	Severity	Location
Optimization	Information	REFLECT.sol L161-162

Description:

The condition `!_isExcluded[sender] && !_isExcluded[recipient]` can be included in `else` .

Recommendation:

The following code can be removed:

```
... else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
    _transferStandard(sender, recipient, amount);
} ...
```

Exhibit-04: Dynamic Rate Between `rSupply` and `tSupply`

Type	Severity	Location
Logical Issue	Informational	

Description:

Suppose the initial total supplies `_tTotal` $= T_0^t$ and `_rTotal` T_0^r, then the initial exchange rate between `rSupply` and `tSupply` $r_0 = T_0^r / T_0^t$.
After we make the first transfer of amount `x` from the initial owner to account `A` , the `r` balance of `A` `_rOwned[A]` $= O_A^r = 0.99x$. And `_rTotal` becomes $T_0^r - 0.01x$ because of the transfer fees.
Then we exclude account `A` such that the `t` balance `_tOwned[A]` $= O_A^r / r_0 = 0.99x T_0^t / T_0^r$.
Now the rate

$$r_1 = \frac{T_0^r - 0.01x - 0.99x}{T_0^t - 0.99x T_0^t / T_0^r} = \frac{T_0^r - x}{T_0^r - 0.99x} \cdot \frac{T_0^r}{T_0^t} < r_0$$

Similarly we can find the exchange rate will decrease as more accounts are excluded. However, as long as the majority of the supply is not excluded, the decrease will be small.

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.