88mph

Security Assessment

February 19th, 2021

For :
88mph

By :
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Angelos Apostolidis @ CertiK
angelos.apostolidis@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	88mph
Description	This audit is centered around the `ZeroCoupon` bond and bond factory contracts of the codebase.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. d814994d7bfdef98a7ab6d9faadf28b034fa2bd6

Audit Summary

Delivery Date	February 19th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	February 1st, 2021 - February 19th, 2021

Vulnerability Summary

Total Issues	4
Total Critical	0
Total Major	0
Total Medium	0
Total Minor	2
Total Informational	2

Executive Summary

We were tasked with auditing the codebase of 88mph and in particular the `ZeroCouponBond` implementation as well as associated contract factory.

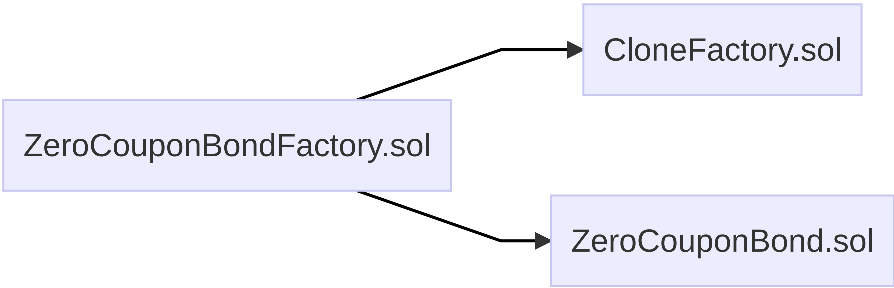
Over the course of the audit we closely inspected the code from a context-agnostic point of view to identify any flaws in the code itself, discounting any and all external interactions it has. As the audit scope was limited to those two files, 88mph contracts the code was interacting with were considered black boxes and correctly behaving for the purposes of the audit.

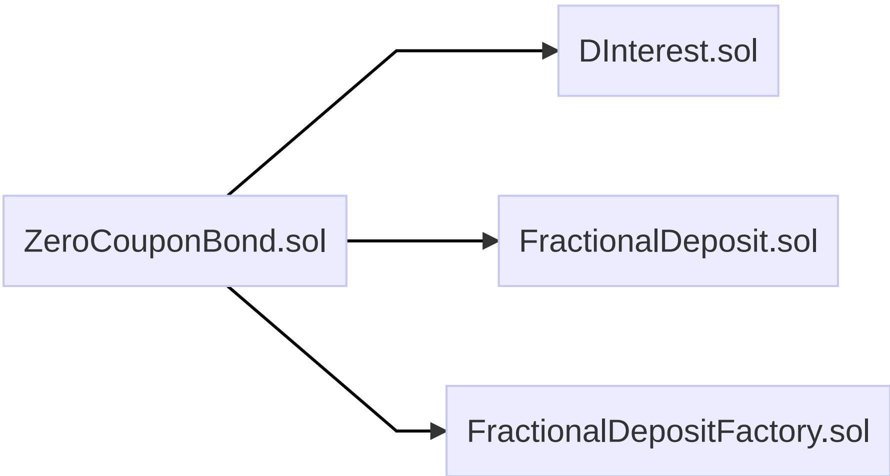
We were unable to identify any prevalent flaws apart from a potentially non-compliant method of validating an ERC20 transfer has occured. The code is of high quality and conforms to the latest security guidelines as well as styling conventions.

Files In Scope

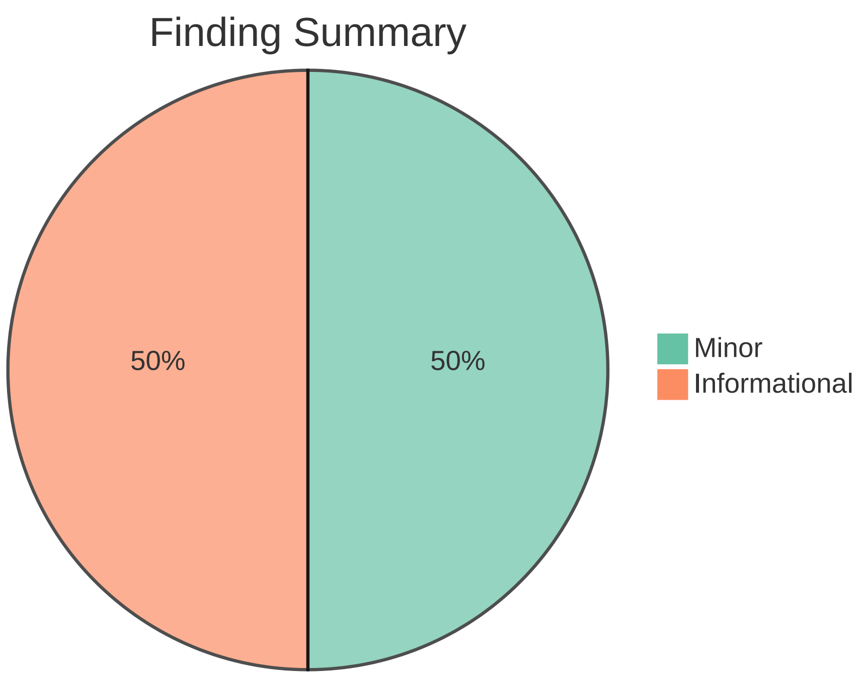
ID	Contract	Location
ZCB	ZeroCouponBond.sol	contracts/fractionals/ZeroCouponBond.sol
ZCF	ZeroCouponBondFactory.sol	contracts/fractionals/ZeroCouponBondFactory.sol

File Dependency Graph





Findings



ID	Title	Type	Severity	Resolved
ZCF-01	Redundant `import` Statement	Gas Optimization	Informational	
ZCB-01	State Layout	Gas Optimization	Informational	
ZCB-02	Inexistant Input Sanitization	Volatile Code	Minor	
ZCB-03	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	Minor	

ZCF-01: Redundant `import` Statement

Type	Severity	Location
Gas Optimization	Informational	ZeroCouponBondFactory.sol L4

Description:

The linked `import` statement is never used throughout the contract.

Recommendation:

We advise to remove redundant code.

Alleviation:

The 88mph development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

ZCB-01: State Layout

Type	Severity	Location
Gas Optimization	Informational	ZeroCouponBond.sol L41-L48

Description:

The state layout should be as tighly packed as possible to 256-bit sized pairs to save gas.

Recommendation:

We advise to change the state layout by putting the `decimals` state variable before the `initialized` one.

Alleviation:

The 88mph development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

ZCB-02: Inexistant Input Sanitization

Type	Severity	Location
Volatile Code	Minor	ZeroCouponBond.sol L62-L91

Description:

The `init()` function does not properly sanitize the input values the user provides.

Recommendation:

We advise to introduce the necessary `require` statements to filter the potential values that could break the flow of the system.

Alleviation:

The 88mph development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

ZCB-03: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	Minor	ZeroCouponBond.sol L115-L119

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true`, many token implementations do not return anything i.e. Tether (USDT). Hence, ensuring that the returned value, in case it exists, is the expected one should be added to the codebase.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked correctly in all circumstances.

Alleviation:

The 88mph development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

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.

Dead Code

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