BasisX

Security Assessment

Jan 16th, 2021

By :
Merkle Capital Management Co., Ltd
roy@merklecap.io

Overview

Project Summary

Project Name	BasisX.io
Description	BasisX Protocol implements a reward board table, where you can stake and withdraw tokens, and a treasury contract, withing which users can swap between BasisX Cash and BasisX Bonds.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository

Audit Summary

Delivery Date	Jan 16th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Dec 22nd, 2020 - Jan 16th, 2021

Vulnerability Summary

Total Issues	9
Total Critical	0
Total Major	0
Total Medium	0
Total Minor	0
Total Informational	9

Executive Summary

The report represents the results of our engagement with the BasisX on their implementation of their reward board table and treasury smart contracts.

Our findings mainly refer to optimizations and Solidity coding standards. Hence, the issues identified pose no threat to the safety of the contract deployement.

Files In Scope

ID	Contract	Location
BOA	Boardroom.sol	contracts/Boardroom.sol
BON	Bond.sol	contracts/Bond.sol
CAS	Cash.sol	contracts/Cash.sol
SHA	Share.sol	contracts/Share.sol
TRE	Treasury.sol	contracts/Treasury.sol

Findings

ID	Title	Type	Severity	Resolved
BOA-01	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	✓
BOA-02	Redundant Variable Initialization	Coding Style	Informational	✓
TRE-01	Unlocked Compiler Version	Language Specific	Informational	✓
TRE-02	Visibility Specifiers Missing	Language Specific	Informational	✓
TRE-03	Ambiguous Variable Assignment	Volatile Code	Informational	✓
TRE-04	Introduction of a `constant` Variable	Gas Optimization	Informational	✓
TRE-05	`if` Block Optimization	Gas Optimization	Informational	✓
TRE-06	Ambiguous Centralization	Control Flow	Informational	✓

BOA-01: Inefficient Greater-Than Comparison w/ Zero

Type	Severity	Location
Gas Optimization	Informational	Boardroom.sol L50, L68, L71, L95, L138, L148

Description:

The linked greater-than comparisons with zero compare variables that are restrained to the non-negative integer range, meaning that the comparator can be changed to an inequality one which is more gas efficient.

Recommendation:

We advise that the above paradigm is applied to the linked greater-than statements.

Alleviation:

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

BOA-02: Redundant Variable Initialization

Type	Severity	Location
Coding Style	Informational	Boardroom.sol L123

Description:

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:

- `uint` / `int` : All `uint` and `int` variable types are initialized at `0`
- `address` : All `address` types are initialized to `address(0)`
- `byte` : All `byte` types are initialized to their `byte(0)` representation
- `bool` : All `bool` types are initialized to `false`
- `ContractType` : All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its
- default value is `ERC20(address(0))`)
  `struct` : All `struct` types are initialized with all their members zeroed out according to this table

Recommendation:

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation:

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

TRE-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Treasury.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

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

TRE-02: Visibility Specifiers Missing

Type	Severity	Location
Language Specific	Informational	Treasury.sol L36, L37

Description:

The linked variable declarations do not have a visibility specifier explicitly set.

Recommendation:

Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We advise that visibility specifiers for the linked variables are explicitly set.

Alleviation:

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

TRE-03: Ambiguous Variable Assignment

Type	Severity	Location
Volatile Code	Informational	Treasury.sol L125

Description:

As per the inline comment, the variable `bondPrice` should be to the price of basisx cash multiplied by the input amount. Yet, the value of the linked variable is only equal to the price of basisx cash.

Recommendation:

We advise the team to either change the variable assignment to match the inline comment, or vice-versa, or remove the variable `bondPrice` if the intended value is equal to that of the variable `cashPrice` , as it would be redundant.

Alleviation:

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

TRE-04: Introduction of a `constant` Variable

Type	Severity	Location
Gas Optimization	Informational	Treasury.sol L38

Description:

The variable `bondDepletionFloor` can be introduced as a `constant` one, as the `constructor` function updates the default data type value to a literal.

Recommendation:

We advise the team to change the mutability of the `bondDepletionFloor` variable to `constant` .

Alleviation:

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

TRE-05: `if` Block Optimization

Type	Severity	Location
Gas Optimization	Informational	Treasury.sol L190-L199

Description:

The statements in the L192 and L197 are redundant and should be executed outside of the `if` block.

Recommendation:

We advise the team to introduce the following statement before the `if` block and remove the old ones:

```
IBasisAsset(cash).mint(address(this), seigniorage);
```

Alleviation:

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

TRE-06: Ambiguous Centralization

Type	Severity	Location
Control Flow	Informational	Treasury.sol L73-L97

Description:

The setter functions give the `owner` total control of the contract.

Recommendation:

We advise the team to revise the linked code block.

Alleviation:

The BasisX 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.