OMS Finance

Cascade

Security Assessment

April 20th, 2021

Audited By:
Sheraz Arshad @ CertiK
sheraz.arshad@certik.org
Reviewed By:
Camden Smallwood @ CertiK
camden.smallwood@certik.org

Disclaimer

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 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	OMS Finance - Cascade
Description	The codebase comprise Staking and Token Pool contracts including Share contract inspired from Reflect Token. Any stake by a user represents their staking share and the users are rewarded with the distribution token based on their staking share.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 5b60d8b0ec30aa6ed91eefd6011fcc5d935131ca 2. 855173d44456e9cf5e21186c9d41dca3f0f12732

Audit Summary

Delivery Date	April 20th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	1
Timeline	April, 5th, 2021 - April 20th, 2021

Vulnerability Summary

Total Issues	13
● **Total Critical**	0
● **Total Major**	0
● **Total Medium**	2
● **Total Minor**	4
● **Total Informational**	7

Executive Summary

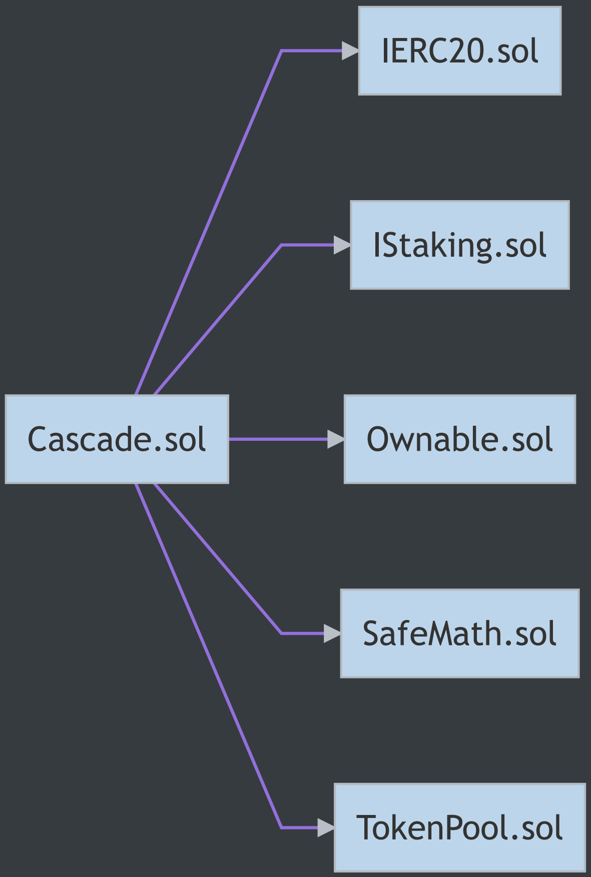
This report represents the results of CertiK's engagement with OMS on their implementation of the Oms Cascade smart contracts.

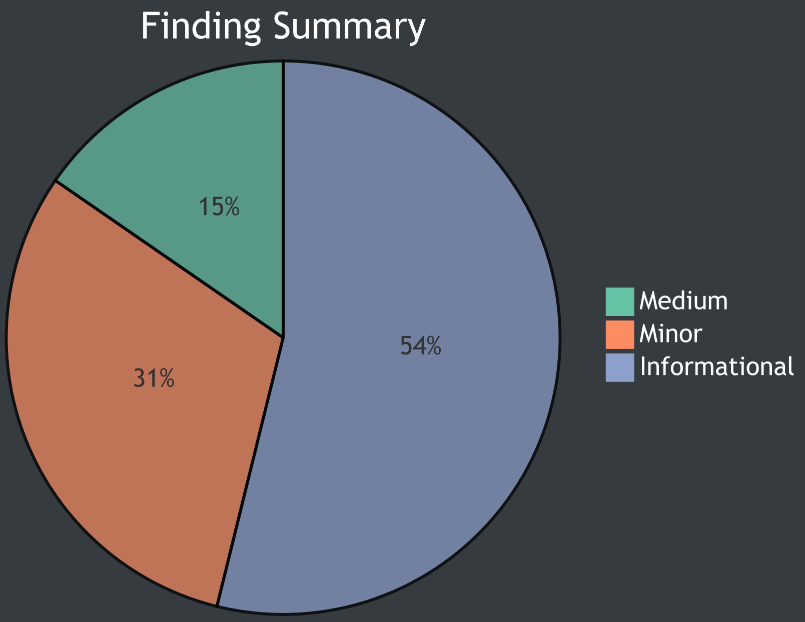
The manual and static analysis were performed in the audit. Our findings mainly refer to optimizations issues, a few minor issues and medium issues. The medium issues comprise the non-checking of addresses that initialize the contracts states against zero address value and minor issues comprise the unsafe `transfer` and `transferFrom` calls to the `ERC20` compatible token contracts. All of the findings are remediated as of commit hash `855173d44456e9cf5e21186c9d41dca3f0f12732`.

Files In Scope

ID	Contract	Location
CAS	Cascade.sol	contracts/v5/Cascade.sol
TPL	TokenPool.sol	contracts/v5/TokenPool.sol
OSE	OmsShare.sol	contracts/v6/OmsShare.sol
OWN	Ownable.sol	contracts/v5/common/Ownable.sol
SMH	SafeMath.sol	contracts/v5/common/SafeMath.sol
IER	IERC20.sol	contracts/v5/interface/IERC20.sol
ISG	IStaking.sol	contracts/v5/interface/IStaking.sol

File Dependency Graph




Finding Summary
15%
31%
54%
Medium
Minor
Informational

Manual Review Findings

ID	Title	Type	Severity	Resolved
CAS-01	Lack of verification for the constructor parameters	Logical Issue	● Medium	✓
CAS-02	Requisite Value of ERC-20 \`transferFrom()\` / \`transfer()\` Call	Logical Issue	● Minor	✓
CAS-03	Requisite Value of ERC-20 \`transferFrom()\` / \`transfer()\` Call	Logical Issue	● Minor	✓
CAS-04	Incorrect condition	Logical Issue	● Minor	✓
CAS-05	Redundant Variable Initialization	Coding Style	● Informational	✓
CAS-06	mappings data can be packed in a struct	Gas Optimization	● Informational	✓
CAS-07	Unnecessary function parameter	Gas Optimization	● Informational	✓
CAS-08	Inefficient storage read	Gas Optimization	● Informational	✓
CAS-09	Unused function parameter	Coding Style	● Informational	✓
CAS-10	Explicitly returning local variable	Gas Optimization	● Informational	✓
TPL-01	Unchecked value of \`transfer\` call	Volatile Code	● Minor	✓
OSE-01	Lack of verification for the constructor parameter	Logical Issue	● Medium	✓
OSE-02	Unlocked Compiler Version	Language Specific	● Informational	✓

CAS-01: Lack of verification for the constructor parameters

Type	Severity	Location
Logical Issue	● Medium	Cascade.sol L85

Description:

The constructor on the aforementioned line does not validate its address type parameters of `stakingToken` and `distributionToken` against zero address value before utilizing them to create `TokenPool` instances. If any of the constructor parameters are passed as zero address then it will result in unwanted state of contract and once set, the `TokenPool` instances will point to zero address as its underlying token.

Recommendation:

We advise to validate the aforementioned address type parameters against zero address value.

Alleviation:

Alleviations were applied as of commit hash `855173d44456e9cf5e21186c9d41dca3f0f12732` .

CAS-02: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	● Minor	Cascade.sol L170

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) leading to unexpected halts in code execution.

Recommendation:

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

Alleviation:

Alleviations were applied as of commit hash `855173d44456e9cf5e21186c9d41dca3f0f12732`.

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

Type	Severity	Location
Logical Issue	🔵 Minor	Cascade.sol L421

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) leading to unexpected halts in code execution.

Recommendation:

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

Alleviation:

Alleviations were applied as of commit hash `855173d44456e9cf5e21186c9d41dca3f0f12732`.

CAS-04: Incorrect condition

Type	Severity	Location
Logical Issue	● Minor	Cascade.sol L401-L402

Description:

The condition inside the `require` statement on the aforementioned line only allows the creation of unlock schedules which are one short of maximum allowed unlock schedules.

Recommendation:

We advise to rectify the condition in `require` statement by substituting the `less-than` comparison with `less-than-or-equal` comparison such that the `require` statement allows creation of unlock schedules equal to maximum allowed unlock schedules.

Alleviation:

Alleviations were applied as of commit hash `855173d44456e9cf5e21186c9d41dca3f0f12732` .

CAS-05: Redundant Variable Initialization

Type	Severity	Location
Coding Style	● Informational	Cascade.sol L27-L28, L33-L38, L432, L438, L469, L215, L217, L221

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:

Alleviations were applied as of commit hash `855173d44456e9cf5e21186c9d41dca3f0f12732` .

CAS-06: mappings data can be packed in a struct

Type	Severity	Location
Gas Optimization	● Informational	Cascade.sol L58, L61

Description:

The mappings on the aforementioned lines have key of type address representing a user's address. These mappings can be combined into a single mapping having address as key type and the value type will be a struct having properties from both of the aforementioned mappings. This will reduce the lookup gas cost when reading data from these mappings.

Recommendation:

We advise to replace the aforementioned mappings with a single mapping by utilizing a struct for the value types across all the aforementioned mappings.

```
struct User {
   UserTotals userTotal;
   Stake[] stakes;
}
```

```
mapping(address => User) private users;
```

Alleviation:

Alleviations were applied as of commit hash `855173d44456e9cf5e21186c9d41dca3f0f12732` .

CAS-07: Unnecessary function parameter

Type	Severity	Location
Gas Optimization	● Informational	Cascade.sol L143

Description:

The function parameter `staker` on the aforementioned line always points to `msg.sender` in the current implementation of the contract. The parameter can be omitted from the function signature and `msg.sender` can be directly used in its place to save gas cost and to increase the legibility of the codebase.

Recommendation:

We recommend to remove the `staker` parameter from the signature of the function.

Alleviation:

Alleviations were applied as of commit hash `855173d44456e9cf5e21186c9d41dca3f0f12732` .

CAS-08: Inefficient storage read

Type	Severity	Location
Gas Optimization	● Informational	Cascade.sol L439

Description:

The conditional on the aforementioned lines repeatedly reads the length of `array` from contract's storage which can be optimized by storing the array's length in a local variable and then utilizing it to save gas cost.

Recommendation:

We advise to assign the `array` length to a local variable instead.

Alleviation:

Alleviations were applied as of commit hash `855173d44456e9cf5e21186c9d41dca3f0f12732`.

CAS-09: Unused function parameter

Type	Severity	Location
Coding Style	● Informational	Cascade.sol L123, L133, L182

Description:

The functions on the aforementioned lines have parameter `data` of type `bytes` which is never used within the aforementioned functions.

Recommendation:

We recommend to remove the `data` parameter from the signatures of the aforementioned functions to increase the legibility of the codebase.

Alleviation:

Alleviations were applied as of commit hash `855173d44456e9cf5e21186c9d41dca3f0f12732`.

CAS-10: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	● Informational	Cascade.sol L462, L200

Description:

The functions on the aforementioned lines explicitly return local variable which increases overall cost of gas.

Recommendation:

Since named return variables can be declared in the signature of a function, consider refactoring to remove the local variable declaration and explicit return statement in order to reduce the overall cost of gas.

Alleviation:

Alleviations were applied as of commit hash `855173d44456e9cf5e21186c9d41dca3f0f12732`.

TPL-01: Unchecked value of `transfer` call

Type	Severity	Location
Volatile Code	🔵 Minor	TokenPool.sol L23

Description:

The linked `transfer()` / `transferFrom()` invocations do not check the return value of the function call which should yield a `true` result in case of a proper ERC-20 implementation.

Recommendation:

As many tokens do not follow the ERC-20 standard faithfully, they may not return a `bool` variable in this function's execution meaning that simply expecting it can cause incompatibility with these types of tokens. Instead, we advise that OpenZeppelin's `SafeERC20.sol` implementation is utilized for interacting with the `transfer()` and `transferFrom()` functions of ERC-20 tokens. The OZ implementation optionally checks for a return value rendering compatible with all ERC-20 token implementations.

Alleviation:

Alleviations were applied as of commit hash `855173d44456e9cf5e21186c9d41dca3f0f12732` .

OSE-01: Lack of verification for the constructor parameter

Type	Severity	Location
Logical Issue	● Medium	OmsShare.sol L29

Description:

The constructor parameter of `reserve_` on the aforementioned line is not validated against zero address value, which, when provided will result in unwanted state of the contract.

Recommendation:

We advise to validate the aforementioned constructor parameter against zero address value.

Alleviation:

Alleviations were applied as of commit hash `855173d44456e9cf5e21186c9d41dca3f0f12732`.

OSE-02: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	OmsShare.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:

Alleviations were applied as of commit hash `855173d44456e9cf5e21186c9d41dca3f0f12732` .

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.

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.

Volatile Code

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

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.