QuillAudits

Audit Report
September, 2021

For

Contents

Scope of the Audit

The scope of this audit was to analyze and document the Ceres Token smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Mythril, Slither, SmartCheck, Surya, Solhint.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.
Low	Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
Informational	These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	0	4
Closed	0	0	0	0

Introduction

During the period of **September 15, 2021, to September 16, 2021** - QuillAudits Team performed a security audit for Ceres smart contracts.

The code for the audit was taken from the following official link:
https://etherscan.io/address/0x1485e9852ac841b52ed44d573036429504f4f602#code

Issues Found

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low level severity issues

No issues were found.

Informational

1. Missing zero address validation

Line	Code
15-26	`constructor( string memory name, string memory symbol, uint8 decimals, address beneficiary, uint256 supply, bytes32 sidechainAssetId) ERC20Detailed(name, symbol, decimals) { _sidechainAssetId = sidechainAssetId; _mint(beneficiary, supply); }`

```
constructor(
        string memory name,
        string memory symbol,
        uint8 decimals,
        address beneficiary,
        uint256 supply,
        bytes32 sidechainAssetId)
        ERC20Detailed(name, symbol, decimals) {
        _sidechainAssetId = sidechainAssetId;
        _mint(beneficiary, supply);

    }
```

Description
The input of address beneficiary should be checked for address(0), otherwise tokens minted to the zero address may be burnt forever.

Remediation
Use a require statement to check for zero address in the constructor.

Status: Acknowledged by the Auditee

2. Floating Pragma

pragma solidity ^0.7.4;

Description
Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation
Lock the pragma version and also consider known bugs for the compiler version that is chosen.

Status: Acknowledged by the Auditee

3. State variables that could be declared immutable

bytes32 public _sidechainAssetId;

Description
The above constant state variable should be declared immutable to save gas.

Remediation
Add the immutable attributes to state variables that never change after contact creation.

Status: Acknowledged by the Auditee

4. Public function that could be declared external

Description
The following public functions that are never called by the contract should be declared external to save gas:
- mintTokens()

Remediation
Use the external attribute for functions never called from the contract.

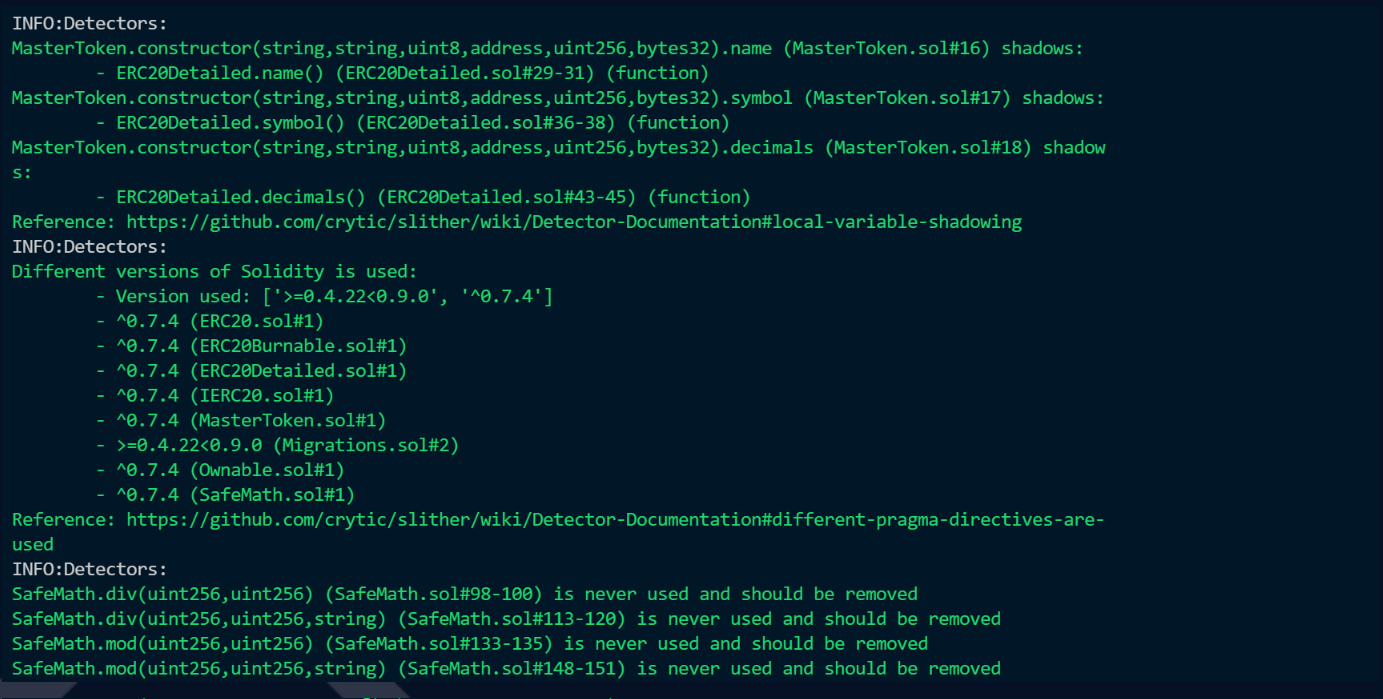
Status: Acknowledged by the Auditee

Functional Tests

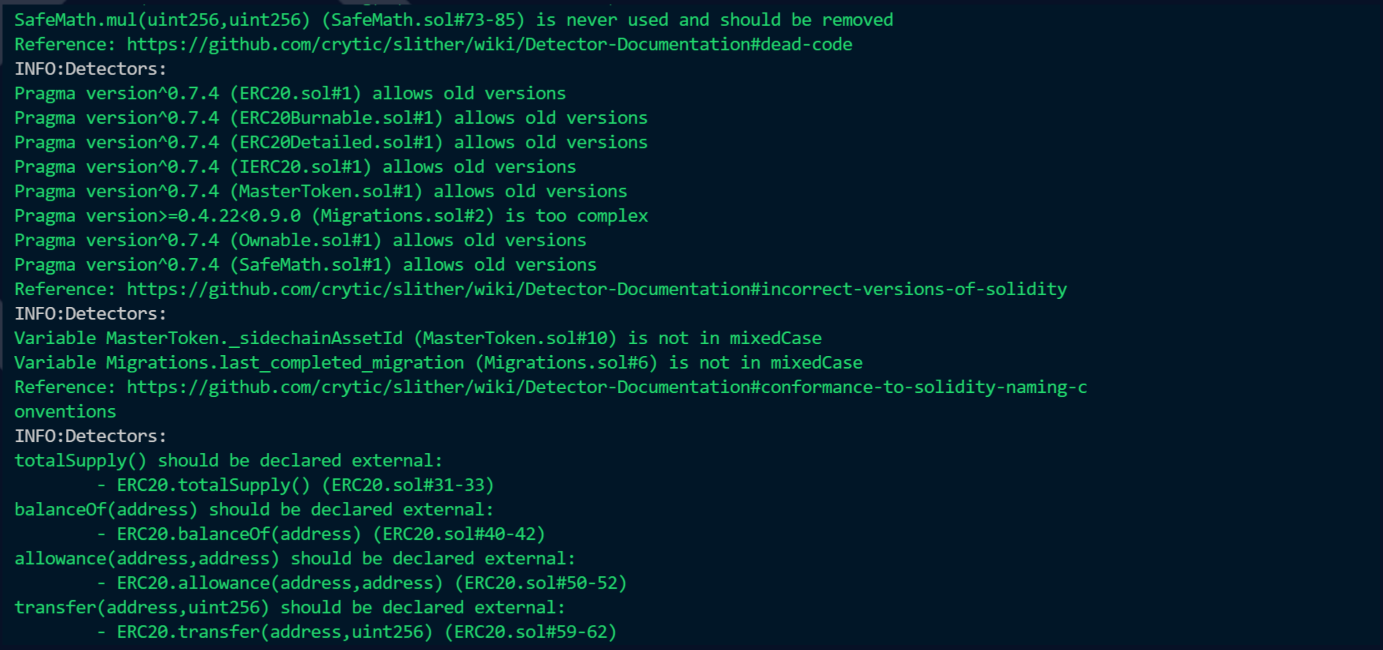
Function Names	Testing results
mintTokens()	Passed

Automated Testing

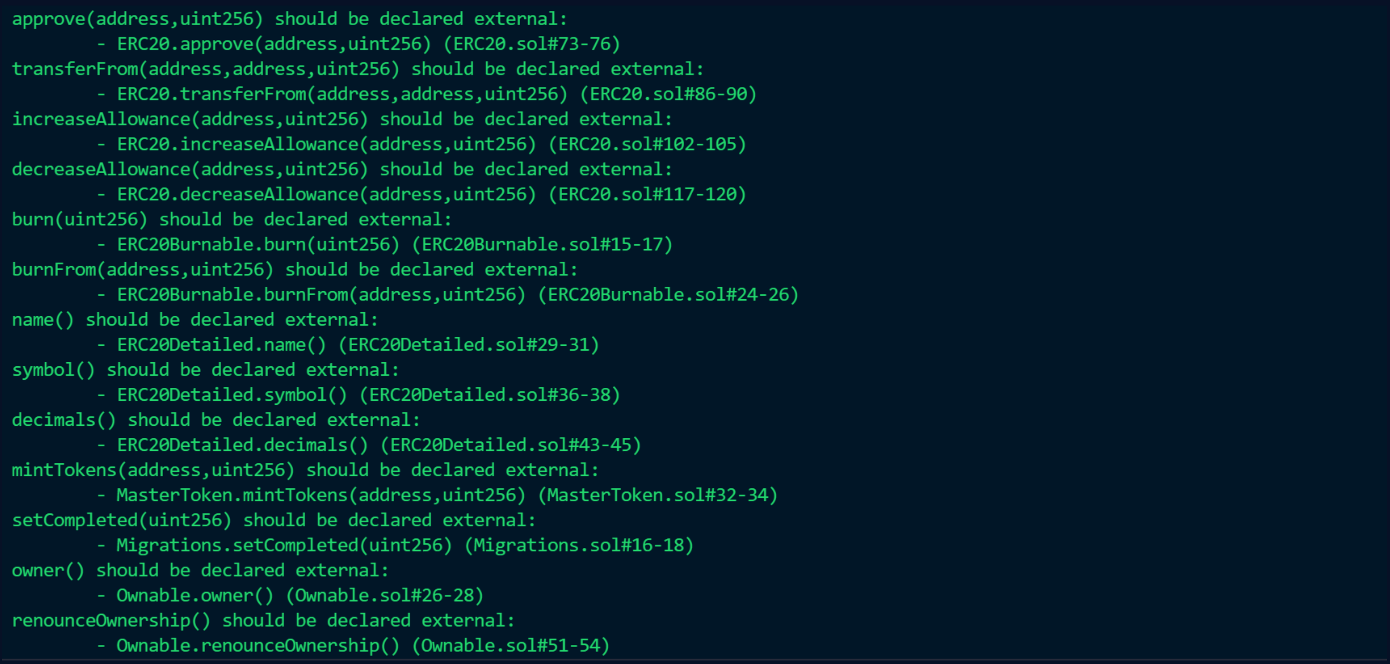
Slither



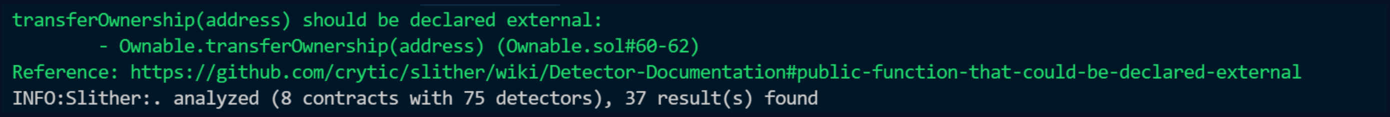
```
INFO:Detectors:
MasterToken.constructor(string,string,uint8,address,uint256,bytes32).name (MasterToken.sol#16) shadows:
        - ERC20Detailed.name() (ERC20Detailed.sol#29-31) (function)
MasterToken.constructor(string,string,uint8,address,uint256,bytes32).symbol (MasterToken.sol#17) shadows:
        - ERC20Detailed.symbol() (ERC20Detailed.sol#36-38) (function)
MasterToken.constructor(string,string,uint8,address,uint256,bytes32).decimals (MasterToken.sol#18) shadow
s:
        - ERC20Detailed.decimals() (ERC20Detailed.sol#43-45) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['>=0.4.22<0.9.0', '^0.7.4']
        - ^0.7.4 (ERC20.sol#1)
        - ^0.7.4 (ERC20Burnable.sol#1)
        - ^0.7.4 (ERC20Detailed.sol#1)
        - ^0.7.4 (IERC20.sol#1)
        - ^0.7.4 (MasterToken.sol#1)
        - >=0.4.22<0.9.0 (Migrations.sol#2)
        - ^0.7.4 (Ownable.sol#1)
        - ^0.7.4 (SafeMath.sol#1)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-
used
INFO:Detectors:
SafeMath.div(uint256,uint256) (SafeMath.sol#98-100) is never used and should be removed
SafeMath.div(uint256,uint256,string) (SafeMath.sol#113-120) is never used and should be removed
SafeMath.mod(uint256,uint256) (SafeMath.sol#133-135) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (SafeMath.sol#148-151) is never used and should be removed
```



```
SafeMath.mul(uint256,uint256) (SafeMath.sol#73-85) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.7.4 (ERC20.sol#1) allows old versions
Pragma version^0.7.4 (ERC20Burnable.sol#1) allows old versions
Pragma version^0.7.4 (ERC20Detailed.sol#1) allows old versions
Pragma version^0.7.4 (IERC20.sol#1) allows old versions
Pragma version^0.7.4 (MasterToken.sol#1) allows old versions
Pragma version>=0.4.22<0.9.0 (Migrations.sol#2) is too complex
Pragma version^0.7.4 (Ownable.sol#1) allows old versions
Pragma version^0.7.4 (SafeMath.sol#1) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable MasterToken._sidechainAssetId (MasterToken.sol#10) is not in mixedCase
Variable Migrations.last_completed_migration (Migrations.sol#6) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Detectors:
totalSupply() should be declared external:
        - ERC20.totalSupply() (ERC20.sol#31-33)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (ERC20.sol#40-42)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (ERC20.sol#50-52)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (ERC20.sol#59-62)
```



```
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (ERC20.sol#73-76)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (ERC20.sol#86-90)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (ERC20.sol#102-105)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (ERC20.sol#117-120)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (ERC20Burnable.sol#15-17)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (ERC20Burnable.sol#24-26)
name() should be declared external:
        - ERC20Detailed.name() (ERC20Detailed.sol#29-31)
symbol() should be declared external:
        - ERC20Detailed.symbol() (ERC20Detailed.sol#36-38)
decimals() should be declared external:
        - ERC20Detailed.decimals() (ERC20Detailed.sol#43-45)
mintTokens(address,uint256) should be declared external:
        - MasterToken.mintTokens(address,uint256) (MasterToken.sol#32-34)
setCompleted(uint256) should be declared external:
        - Migrations.setCompleted(uint256) (Migrations.sol#16-18)
owner() should be declared external:
        - Ownable.owner() (Ownable.sol#26-28)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (Ownable.sol#51-54)
```



```
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (Ownable.sol#60-62)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:. analyzed (8 contracts with 75 detectors), 37 result(s) found
```

Description

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract, but relying on other contracts might cause Reentrancy Vulnerability. It has to be noted that the token has no fixed supply, and the contract owner has permission to mint new tokens.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Ceres platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Ceres Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
September, 2021

For

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com