QuillAudits

Audit Report
October, 2022

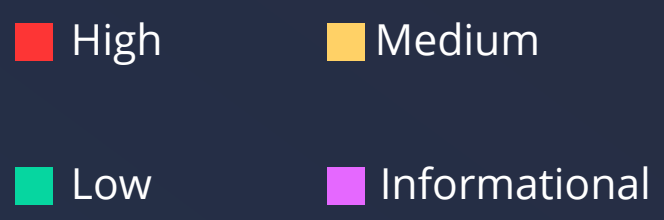
For

AssetMantle

Table of Content

Executive Summary

Project Name	AssetMantle
Overview	CosmosERC20.sol is a token with only ERC20 functionality. It inherits the OpenZeppelin ERC20 library and sub-libraries. There are no additional functions in the codebase.
Timeline	27 September, 2022 to 12 october, 2022.
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse AssetMantle codebase for quality, security, and correctness. https://etherscan.io/address/0x2c4f1df9c7de0c59778936c9b145ff56813f3295#code



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	4
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

Types of Severities

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 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.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

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

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - Cosmos Token

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

A.1: No way to deal with unexpected ethers sent into the contract

Description

The cosmosERC20.sol contract does not have a fallback or receive function to deal with misconfigurations or transfers made out of place. Even if there is no need to receive tokens or ethers, the fallback or receive function can be made to revert, but this does not entirely remove the possibility of an unexpected increase in contract balances.

Remediation

It is best practice to include a fallback or receive function in the contract.

Status

Acknowledged

A.2: Unlocked pragma (pragma solidity ^0.8.0)

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

Here all but one of the in-scope contracts have an unlocked pragma, it is recommended to lock all.

Status

Acknowledged

A3. MAX_UINT variable declared

Description

The variable MAX_UINT is only hard coded into the contract and as such can be made constant to save gas.

Remediation

Declare the MAX_UINT variable as constant.

Status

Acknowledged

A.4: cosmosDecimals variable size

Description

The variable cosmosDecimals is declared as uint8 but can be explicitly made a uint256 since other storage slots are 256bits. There is no storage optimization here as of now, possibly with the increase in complexity of the contracts there would be rearrangement as required.

Remediation

Make the cosmosDecimals variable uint256 instead of uint8.

Status

Acknowledged

Functional Testing

Cosmos Token

- ✓ should return the name, symbol and totalSupply of token
- ✓ should return the custom decimals passed
- ✓ should implement ERC20 functionality (e.g. transfer, approve)

Automated Tests

```
CosmosERC20.constructor(address,string,string,uint8).name (contracts/CosmosToken.sol#27) shadows:
        - ERC20._name (contracts/ERC20.sol#41) (state variable)
CosmosERC20.constructor(address,string,string,uint8).symbol (contracts/CosmosToken.sol#28) shadows:
        - ERC20._symbol (contracts/ERC20.sol#42) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

CosmosERC20.constructor(address,string,string,uint8).gravityAddress (contracts/CosmosToken.sol#28) lacks a zero-check on :
                - gravityAddress = _gravityAddress (contracts/CosmosToken.sol#32)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.1']
        - ^0.8.0 (contracts/Context.sol#3)
        - ^0.8.1 (contracts/CosmosToken.sol#3)
        - ^0.8.0 (contracts/ERC20.sol#3)
        - ^0.8.0 (contracts/IERC20.sol#3)
        - ^0.8.0 (contracts/IERC20Metadata.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Context._msgData() (contracts/Context.sol#20-22) is never used and should be removed
ERC20._burn(address,uint256) (contracts/ERC20.sol#274-288) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (contracts/Context.sol#3) allows old versions
Pragma version^0.8.1 (contracts/CosmosToken.sol#3) allows old versions
Pragma version^0.8.0 (contracts/ERC20.sol#3) allows old versions
Pragma version^0.8.0 (contracts/IERC20Metadata.sol#4) allows old versions
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Variable CosmosERC20.MAX_UINT (contracts/CosmosToken.sol#8) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

CosmosERC20.MAX_UINT (contracts/CosmosToken.sol#8) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

decimals() should be declared external:
        - CosmosERC20.decimals() (contracts/CosmosToken.sol#32-34)
        - ERC20.decimals() (contracts/ERC20.sol#90-98)
totalSupply() should be declared external:
        - CosmosERC20.totalSupply() (contracts/CosmosToken.sol#45-49)
        - ERC20.totalSupply() (contracts/ERC20.sol#103-105)
name() should be declared external:
        - ERC20.name() (contracts/ERC20.sol#61-63)
symbol() should be declared external:
        - ERC20.symbol() (contracts/ERC20.sol#69-71)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (contracts/ERC20.sol#112-115)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (contracts/ERC20.sol#120-122)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (contracts/ERC20.sol#133-134)
transferFrom(address,address,uint256) should be declared external:
```

```
        - ERC20.transferFrom(address,address,uint256) (contracts/ERC20.sol#149-161)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (contracts/ERC20.sol#177-180)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (contracts/ERC20.sol#196-204)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Closing Summary

In this report, we have considered the security of the AssetMantle. We performed our audit according to the procedure described above.

Some issues of Low and Informational severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the AssetMantle 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 AssetMantle Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

600+
Audits Completed

$15B
Secured

600K
Lines of Code Audited

Follow Our Journey

Audit Report
October, 2022

For

AssetMantle

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com