QuillAudits

Audit Report
July, 2021

Contents

Scope of Audit

The scope of this audit was to analyze and document the Catharsis 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 analyzed 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.
SmartCheck.

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 security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of 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 possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

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

High severity issues

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 to be fixed before moving to a live environment.

Medium level severity issues

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 level severity issues

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 four issues which 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
Closed	0	0	2	5

Introduction

During the period of **July 12, 2021 to July 15, 2021** - QuillAudits Team performed a security audit for Catharsis smart contracts.

The code for the audit was taken from the following official link: https://github.com/CatharsisNetwork/catharsis-vesting/blob/main/contracts/Vesting.sol

Note	Date	Commit hash
Version 1	July	22d1bb2e234b79b10c33a4ec76ce2a98d2825364
Version 2	July	e16459f6f7b348a89ac61dadbb59e4cb0f455be2
Version 3	July	de7e905754ac582266f96386e15cb1a7f3f93b69

Issues Found – Code Review / Manual Testing

High severity issues

No issues were found.

Medium severity issues

Low level severity issues

1. Wrong inputs can be passed to locks

Line	Code

144-156	for (i; i < inputsLen; i++) {
	lockLen = _input[i].unlockAt.length;
	for (ii; ii < lockLen; ii++) {
	if (_input[i].account == address(0)) {
	require(false, "Zero address");
	} else if (
	_input[i].unlockAt.length != _input[i].amounts.length ||
	_input[i].unlockAt.length > MAX_LOCK_LENGTH
	) {
	require(false, "Wrong array length");
	} else if (_input[i].unlockAt.length == 0) {
	require(false, "Zero array length");
	}

Description

The function lockBatch() can be passed an empty array as input for unlockAt[] in the lock. This means the nested for loop is never executed, and the important requirements inside it are never checked. Now, this allows the role Owner to input:

address(0) as account
Empty array [] as unlockAt[]
Array with any number of elements (including 0) as amounts[]

If such input is passed, the following functions will revert :

pendingReward()
claim()

Remediation
As the function is protected by the modifier onlyOwner, allowing only the role Owner to call this function, the inputs should be checked properly before calling the function by the owner.
We also recommend checking that if the input array is empty for a lock, then do not push that lock into the _balance[account] mapping.

Status: Closed
Function lockBatch() was fixed in Version 2, and now the security checks are done before saving the locks into the storage.

2. Wrong inputs can be passed to locks

Line	Code
197	if (ii == l - ii) {

Description
In the lockBatch() function, TokenVested event is emitted for every new lock with the amount and address as parameters. Before emitting the event, the amount is calculated using a for loop. But the event is emitted before completion of that loop, with wrong values of amount.
This is due to an error in the statement which is used to determine the end of the loop.

Remediation
Use `l - 1` in place of `l - ii` , to determine the end of the loop.

Status: Closed
This issue was fixed in version 3.

Informational

3. Missing Events for Significant Transactions

Description
The missing event makes it difficult to track off-chain liquidity fee changes. An event should be emitted for significant transactions calling the function lock() and lockBatch().

Remediation
We recommend emitting the unused event TokensVested when the lock() and lockBatch() functions are called.

Status: Closed
In Version 2, Event 'TokensVested' is emitted for lock() and lockBatch() functions.

4. Redundant Code

Line	Code

154-156	} else if (_input[i].unlockAt.length == 0) {
	require(false, "Zero array length");
	}

Description
This statement is inside a for loop, which iterates over unlockAt[] array. If the array has length == 0, then the loop is never executed, and this condition is never checked.

Remediation
We recommend removing this code.

Status: Closed
This issue was fixed in Version 2.

5. Floating Pragma

pragma solidity ^0.8.4;

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: Closed
In Version 2, the solidity pragma version was locked to 0.8.4.

6. State variables that could be declared immutable

uint256 public startAt
IERC20 public token

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: Closed
The variables were declared immutable in version 3.

7. 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:

getNextUnlock()
getNextUnlockByIndex()
getLocks()
getLocksLength()

Remediation

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

Status: Closed

Functions were declared as external in Version 2.

Functional test

Function Names	Testing results
lock()	Passed
lockBatch()	Passed
getNextUnlock()	Passed
getNextUnlockByIndex()	Passed
pendingReward()	Passed
pendingRewardInRange()	Passed
claim()	Passed
claimInRange()	Passed
getLocksLength()	Passed
getItemsLengthByLockIndex()	Passed
getLocks()	Passed

Automated Testing

Slither

```
INFO:Detectors:
Vesting._pendingReward(address,uint256,uint256).amount (Vesting.sol#258) is a local variable never initialized
Vesting._claim(address,uint256,uint256).toRelease (Vesting.sol#288) is a local variable never initialized
Vesting.lockBatch(Vesting.LockBatchInput[]).i (Vesting.sol#136) is a local variable never initialized
Vesting._claim(address,uint256,uint256).amount (Vesting.sol#283) is a local variable never initialized
Vesting._getNextUnlock(address,uint256).i (Vesting.sol#322) is a local variable never initialized
Vesting.getNextUnlock(address).i (Vesting.sol#187) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
AccessControlEnumerable.grantRole(bytes32,address) (@openzeppelin\contracts\access\AccessControlEnumerable.sol#59-62) ignores return va
lue by _roleMembers[role].add(account) (@openzeppelin\contracts\access\AccessControlEnumerable.sol#61)
AccessControlEnumerable.revokeRole(bytes32,address) (@openzeppelin\contracts\access\AccessControlEnumerable.sol#67-70) ignores return v
alue by _roleMembers[role].remove(account) (@openzeppelin\contracts\access\AccessControlEnumerable.sol#69)
AccessControlEnumerable.renounceRole(bytes32,address) (@openzeppelin\contracts\access\AccessControlEnumerable.sol#75-78) ignores return
 value by _roleMembers[role].remove(account) (@openzeppelin\contracts\access\AccessControlEnumerable.sol#77)
AccessControlEnumerable._setupRole(bytes32,address) (@openzeppelin\contracts\access\AccessControlEnumerable.sol#83-86) ignores return v
alue by _roleMembers[role].add(account) (@openzeppelin\contracts\access\AccessControlEnumerable.sol#85)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
ERC20Mock.constructor(string,string)._name (ERC20Mock.sol#7) shadows:
        - ERC20._name (@openzeppelin\contracts\token\ERC20\ERC20.sol#40) (state variable)
ERC20Mock.constructor(string,string)._symbol (ERC20Mock.sol#7) shadows:
        - ERC20._symbol (@openzeppelin\contracts\token\ERC20\ERC20.sol#41) (state variable)
ERC20PresetMinterPauser.constructor(string,string).name (@openzeppelin\contracts\token\ERC20\presets\ERC20PresetMinterPauser.sol#35) sh
adows:
        - ERC20.name() (@openzeppelin\contracts\token\ERC20\ERC20.sol#60-62) (function)
        - IERC20Metadata.name() (@openzeppelin\contracts\token\ERC20\extensions\IERC20Metadata.sol#16) (function)
ERC20PresetMinterPauser.constructor(string,string).symbol (@openzeppelin\contracts\token\ERC20\presets\ERC20PresetMinterPauser.sol#35)
shadows:
        - ERC20.symbol() (@openzeppelin\contracts\token\ERC20\ERC20.sol#68-70) (function)
        - IERC20Metadata.symbol() (@openzeppelin\contracts\token\ERC20\extensions\IERC20Metadata.sol#21) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Reentrancy in Vesting.lock(address,uint256[],uint256[]) (Vesting.sol#88-122):
        External calls:
        - token.safeTransferFrom(msg.sender,address(this),totalAmount) (Vesting.sol#115)
        State variables written after the call(s):
        - _balances[_account].locks.push(Lock(_amounts,_unlockAt,0)) (Vesting.sol#117-121)
Reentrancy in Vesting.lockBatch(Vesting.LockBatchInput[]) (Vesting.sol#128-178):
        External calls:
        - token.safeTransferFrom(msg.sender,address(this),totalAmount) (Vesting.sol#168)
        State variables written after the call(s):
        - _balances[_input[i].account].locks.push(Lock(_input[i].amounts,_input[i].unlockAt,0)) (Vesting.sol#172-176)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in Vesting._claim(address,uint256,uint256) (Vesting.sol#279-313):
        External calls:
        - token.safeTransfer(_participant,claimed) (Vesting.sol#311)
        Event emitted after the call(s):
        - TokensClaimed(_participant,claimed) (Vesting.sol#312)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
```

```
Vesting._pendingReward(address,uint256,uint256) (Vesting.sol#253-277) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp >= _balances[_participant].locks[i].unlockAt[ii] (Vesting.sol#265)
Vesting._claim(address,uint256,uint256) (Vesting.sol#279-313) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp >= _balances[_participant].locks[i].unlockAt[ii] (Vesting.sol#291)
Vesting._getNextUnlock(address,uint256) (Vesting.sol#315-329) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < _lock.unlockAt[i] (Vesting.sol#324)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (@openzeppelin\contracts\utils\Address.sol#26-36) uses assembly
        - INLINE ASM (@openzeppelin\contracts\utils\Address.sol#32-34)
Address._verifyCallResult(bool,bytes,string) (@openzeppelin\contracts\utils\Address.sol#189-209) uses assembly
        - INLINE ASM (@openzeppelin\contracts\utils\Address.sol#201-204)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
INFO:Detectors:
Pragma version^0.8.0 (@openzeppelin\contracts\access\AccessControl.sol#3) necessitates a version too recent to be trusted. Consider dep
loying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\access\AccessControlEnumerable.sol#3) necessitates a version too recent to be trusted. Co
nsider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\utils\Address.sol#3) necessitates a version too recent to be trusted. Consider deploying
with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\utils\Context.sol#3) necessitates a version too recent to be trusted. Consider deploying
with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\utils\structs\EnumerableSet.sol#3) necessitates a version too recent to be trusted. Consi
der deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\utils\introspection\ERC165.sol#3) necessitates a version too recent to be trusted. Consid
er deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\token\ERC20\ERC20.sol#3) necessitates a version too recent to be trusted. Consider deploy
ing with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\token\ERC20\extensions\ERC20Burnable.sol#3) necessitates a version too recent to be trust
ed. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (ERC20Mock.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\token\ERC20\extensions\ERC20Pausable.sol#3) necessitates a version too recent to be trust
ed. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\token\ERC20\presets\ERC20PresetMinterPauser.sol#3) necessitates a version too recent to b
e trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\utils\introspection\IERC165.sol#3) necessitates a version too recent to be trusted. Consi
der deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\token\ERC20\IERC20.sol#3) necessitates a version too recent to be trusted. Consider deplo
ying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\token\ERC20\extensions\IERC20Metadata.sol#3) necessitates a version too recent to be trus
```

```
ted. Consider deploying with 0.6.12/0.7.6
Pragma version>=0.4.22<0.9.0 (Migrations.sol#2) is too complex
Pragma version^0.8.0 (@openzeppelin\contracts\access\Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying
 with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\security\Pausable.sol#3) necessitates a version too recent to be trusted. Consider deploy
ing with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\security\ReentrancyGuard.sol#3) necessitates a version too recent to be trusted. Consider
 deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\token\ERC20\utils\SafeERC20.sol#3) necessitates a version too recent to be trusted. Consi
der deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin\contracts\utils\Strings.sol#3) necessitates a version too recent to be trusted. Consider deploying
with 0.6.12/0.7.6
Pragma version^0.8.4 (Vesting.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (@openzeppelin\contracts\utils\Address.sol#54-59):
        - (success) = recipient.call{value: amount}() (@openzeppelin\contracts\utils\Address.sol#57)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (@openzeppelin\contracts\utils\Address.sol#122-133):
        - (success,returndata) = target.call{value: value}(data) (@openzeppelin\contracts\utils\Address.sol#131)
Low level call in Address.functionStaticCall(address,bytes,string) (@openzeppelin\contracts\utils\Address.sol#151-160):
        - (success,returndata) = target.staticcall(data) (@openzeppelin\contracts\utils\Address.sol#158)
Low level call in Address.functionDelegateCall(address,bytes,string) (@openzeppelin\contracts\utils\Address.sol#178-187):
        - (success,returndata) = target.delegatecall(data) (@openzeppelin\contracts\utils\Address.sol#185)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
INFO:Detectors:
Variable Migrations.last_completed_migration (Migrations.sol#6) is not in mixedCase
Parameter Vesting.getLocks(address,uint256)._participant (Vesting.sol#52) is not in mixedCase
Parameter Vesting.getLocks(address,uint256)._index (Vesting.sol#52) is not in mixedCase
Parameter Vesting.getLocksLength(address)._participant (Vesting.sol#68) is not in mixedCase
Parameter Vesting.getItemsLengthByLockIndex(address,uint256)._participant (Vesting.sol#75) is not in mixedCase
Parameter Vesting.getItemsLengthByLockIndex(address,uint256)._lockIndex (Vesting.sol#75) is not in mixedCase
Parameter Vesting.lock(address,uint256[],uint256[])._account (Vesting.sol#88) is not in mixedCase
Parameter Vesting.lock(address,uint256[],uint256[])._unlockAt (Vesting.sol#88) is not in mixedCase
Parameter Vesting.lock(address,uint256[],uint256[])._amounts (Vesting.sol#88) is not in mixedCase
Parameter Vesting.lockBatch(Vesting.LockBatchInput[])._input (Vesting.sol#128) is not in mixedCase
Parameter Vesting.getNextUnlock(address)._participant (Vesting.sol#184) is not in mixedCase
Parameter Vesting.getNextUnlockByIndex(address,uint256)._participant (Vesting.sol#205) is not in mixedCase
Parameter Vesting.getNextUnlockByIndex(address,uint256)._lockIndex (Vesting.sol#205) is not in mixedCase
Parameter Vesting.pendingReward(address)._participant (Vesting.sol#220) is not in mixedCase
Parameter Vesting.pendingRewardInRange(address,uint256,uint256)._participant (Vesting.sol#227) is not in mixedCase
Parameter Vesting.pendingRewardInRange(address,uint256,uint256)._from (Vesting.sol#227) is not in mixedCase
Parameter Vesting.pendingRewardInRange(address,uint256,uint256)._to (Vesting.sol#227) is not in mixedCase
Parameter Vesting.claim(address)._participant (Vesting.sol#238) is not in mixedCase
Parameter Vesting.claimInRange(address,uint256,uint256)._participant (Vesting.sol#245) is not in mixedCase
Parameter Vesting.claimInRange(address,uint256,uint256)._from (Vesting.sol#245) is not in mixedCase
Parameter Vesting.claimInRange(address,uint256,uint256)._to (Vesting.sol#245) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "i (Vesting.sol#138)" inVesting (Vesting.sol#15-331)
Redundant expression "ii (Vesting.sol#140)" inVesting (Vesting.sol#15-331)
Redundant expression "i (Vesting.sol#171)" inVesting (Vesting.sol#15-331)
```

```
Redundant expression "i (Vesting.sol#188)" inVesting (Vesting.sol#15-331)
Redundant expression "i (Vesting.sol#262)" inVesting (Vesting.sol#15-331)
Redundant expression "ii (Vesting.sol#264)" inVesting (Vesting.sol#15-331)
Redundant expression "i (Vesting.sol#287)" inVesting (Vesting.sol#15-331)
Redundant expression "ii (Vesting.sol#290)" inVesting (Vesting.sol#15-331)
Redundant expression "i (Vesting.sol#323)" inVesting (Vesting.sol#15-331)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
getRoleMember(bytes32,uint256) should be declared external:
        - AccessControlEnumerable.getRoleMember(bytes32,uint256) (@openzeppelin\contracts\access\AccessControlEnumerable.sol#44-46)
getRoleMemberCount(bytes32) should be declared external:
        - AccessControlEnumerable.getRoleMemberCount(bytes32) (@openzeppelin\contracts\access\AccessControlEnumerable.sol#52-54)
name() should be declared external:
        - ERC20.name() (@openzeppelin\contracts\token\ERC20\ERC20.sol#60-62)
symbol() should be declared external:
        - ERC20.symbol() (@openzeppelin\contracts\token\ERC20\ERC20.sol#68-70)
decimals() should be declared external:
        - ERC20.decimals() (@openzeppelin\contracts\token\ERC20\ERC20.sol#85-87)
totalSupply() should be declared external:
        - ERC20.totalSupply() (@openzeppelin\contracts\token\ERC20\ERC20.sol#92-94)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (@openzeppelin\contracts\token\ERC20\ERC20.sol#99-101)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (@openzeppelin\contracts\token\ERC20\ERC20.sol#111-114)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (@openzeppelin\contracts\token\ERC20\ERC20.sol#130-133)
```

```
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (@openzeppelin\contracts\token\ERC20\ERC20.sol#148-162)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (@openzeppelin\contracts\token\ERC20\ERC20.sol#176-179)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (@openzeppelin\contracts\token\ERC20\ERC20.sol#195-203)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (@openzeppelin\contracts\token\ERC20\extensions\ERC20Burnable.sol#19-21)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (@openzeppelin\contracts\token\ERC20\extensions\ERC20Burnable.sol#34-41)
mint(address,uint256) should be declared external:
        - ERC20PresetMinterPauser.mint(address,uint256) (@openzeppelin\contracts\token\ERC20\presets\ERC20PresetMinterPauser.sol#51-54)
pause() should be declared external:
        - ERC20PresetMinterPauser.pause() (@openzeppelin\contracts\token\ERC20\presets\ERC20PresetMinterPauser.sol#65-68)
unpause() should be declared external:
        - ERC20PresetMinterPauser.unpause() (@openzeppelin\contracts\token\ERC20\presets\ERC20PresetMinterPauser.sol#79-82)
setCompleted(uint256) should be declared external:
        - Migrations.setCompleted(uint256) (Migrations.sol#16-18)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (@openzeppelin\contracts\access\Ownable.sol#53-55)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (@openzeppelin\contracts\access\Ownable.sol#61-64)
getLocks(address,uint256) should be declared external:
        - Vesting.getLocks(address,uint256) (Vesting.sol#52-63)
getLocksLength(address) should be declared external:
        - Vesting.getLocksLength(address) (Vesting.sol#68-70)
```

```
getNextUnlock(address) should be declared external:
        - Vesting.getNextUnlock(address) (Vesting.sol#184-200)
getNextUnlockByIndex(address,uint256) should be declared external:
        - Vesting.getNextUnlockByIndex(address,uint256) (Vesting.sol#205-215)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:. analyzed (23 contracts with 75 detectors), 103 result(s) found
```

Results

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.

Disclaimer

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

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.

Numerous issues were discovered during the initial audit. All of them are now fixed and checked for correctness.

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com