QuillAudits

Audit Report
March, 2022

For

ritestream

Contents

Contents

Scope of the Audit

The scope of this audit was to analyze and document the Ritestream 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
Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis, Theo.

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 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	1	1	1
Closed	1	2	6	8

Introduction

From **March 7, 2022** to **March 23, 2022** - QuillAudits Team performed a security audit for Ritestream smart contracts.

The code for the audit was taken from following the official link:
https://github.com/ritestream/ritestream-contract

Fixed In:

V	Date	Commit ID	Date
1	7th March	c2124bd754b9cf5bc4238cda731cdefa013dac12	contracts/*
2	21st March	9403ffa29fc7fa32b649496a90fa2a7404ce6d5e	contracts/*

Issues Found – Code Review / Manual Testing

A. SaleVesting Contract

High severity issues

No issues were found.

Medium severity issues

A.1 Missing appropriate value checks for _TGEDate

Line	Function - constructor()

36	constructor(address _RITE, uint256 _TGEDate) {
	self = address(this);
	RITE = _RITE;
	TGEDate = _TGEDate;
	}

Description

There is missing appropriate value checks for _TGEDate parameter in constructor. Consider adding at least a zero value check for _TGEDate in the constructor. Right now even _TGEDate can be set to a value that is in past or does not correspond to epoch time.

Remediation

Consider adding appropriate require checks for the parameter such as the require check in setTGEDate() function.

Status: Closed

Low severity issues

A.2 Missing zero address validation

Line	Function - constructor()

36	constructor(address _RITE, uint256 _TGEDate) {
	self = address(this);
	RITE = _RITE;
	TGEDate = _TGEDate;
	}

Description
Missing zero address check for _RITE.

Remediation
It is advised to add a require check for the same.

Status: Closed

A.3 For loop over dynamic array

Line	Code/Function
60	for (uint256 i = 0; i < count; i++) {

Description
For loop exists over dynamic array on line: 60. Programming patterns that are harmless in centralized applications can lead to Denial of Service conditions in smart contracts when the cost of executing a function exceeds the block gas limit.

Remediation
Consider adding a require check on count variable to keep a check on the maximum size over which the for loop will run over.
Refer - https://swcregistry.io/docs/SWC-128

Status: Closed

A.4 No check for Vesting tokens

Line	Code/Function
48	function setVesting(VestingDetail[] calldata _vestingDetails) external onlyOwner

Description
It would be a security best practice to simultaneously check whether the smart contract holds the vestingAmount of tokens during the execution of setVesting() function when the vesting schedules are being set.

Remediation
Consider adding the required code for the same.

Status: Acknowledged

Informational Issues

A.5 Missing events for critical function

Line	Code/Function

198	function setTGEDate(uint256 _date) external onlyOwner {
	require(_date > block.timestamp, "TGE date is not valid");
	TGEDate = _date;
	}

Description
Missing event for setTGEDate() function.

Remediation
Add and emit event for setTGEDate() function as it is a critical operation.
This would be a best practice for offchain monitoring.

Status: Closed

A.6 Comparison with a boolean constant

Line	Code/Function
94	require(_vestingDetails[i].initialClaimed == false, "Initial claimed is not valid");

Line	Code/Function
144	if (vestingDetails[beneficiary].initialClaimed == false && vestingDetails[beneficiary].initialAmount > 0)

Description

Boolean constants can be used directly and do not need to be compare to true or false.

Remediation

You can use directly the boolean constant such as !vestingDetails[beneficiary].initialClaimed

Refer- https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Status: Closed

B. TeamVesting contract

High severity issues

No issues were found.

Medium severity issues

B.1 Missing appropriate value checks for _startDate

Line	Function - constructor()

38	constructor(address _RITE, uint256 _startDate) {
	self = address(this);
	RITE = _RITE;
	//Vesting start date
	startDate = _startDate;
	}

Description

startDate can be set to any arbitrary value such as 0 or any date of the past.

Remediation

It is advised to add sufficient require checks for the same.

Status: Closed

Low severity issues

B.2 Missing zero address check

Line	Code/Function

38	constructor(address _RITE, uint256 _startDate) {
	self = address(this);
	RITE = _RITE;
	//Vesting start date
	startDate = _startDate;
	}

Line	Code/Function
48	function setVesting(VestingDetail[] calldata _vestingDetails) external onlyOwner

Description

There is missing zero address check for _RITE in constructor and beneficiary address in the setVesting() function

Remediation

Consider adding a require check for the same

Status: Closed

B.3 For loop over dynamic array

Line	Code/Function

59	for (uint256 i = 0; i < count; i++) {
	address beneficiary = _vestingDetails[i].beneficiary;
	//Check if beneficiary already has a vesting

Description

For loop over dynamic array on line: 59. Programming patterns that are harmless in centralized applications can lead to Denial of Service conditions in smart contracts when the cost of executing a function exceeds the block gas limit.

Remediation

Consider adding a require check to keep a check on the maximum size over which the for loop will run over.

Refer- https://swcregistry.io/docs/SWC-128

Status: Closed

B.4 Usage of block.timestamp

Line	Code/Function
81	crequire(_vestingDetails[i].claimStartTime > block.timestamp, "Beneficiary has no claimStartTime");

Line	Code/Function
178	vestingDetails[beneficiary].lastClaimedTime = block.timestamp;

Description

On line: 82 and 178 block.timestamp has been used. It should be noted that block.timestamp can give you a sense of the current time or a time delta, however, they are not safe to use for most purposes. It can be manipulated and altered by miners by upto 15 seconds.

Remediation

Usage of block.timestamp can lead to unexpected results. It is advised to use time based oracles instead.

Status: Acknowledged

Informational Issues

B.5 Comparison with a boolean constant

Line	Code/Function
101	require(_vestingDetails[i].initialClaimed == false, "Initial claimed can not be true");
132	require(vestingDetails[beneficiary].exists == true, "Beneficiary has terminated");

Line	Code/Function
150	if (vestingDetails[beneficiary].initialClaimed == false && vestingDetails[beneficiary].initialAmount > 0)
204	require(vestingDetails[beneficiary].exists == true, "Beneficiary is already terminated");

Description

There is comparison with boolean on lines 102, 133, 151 and 205

Remediation

Comparison with a boolean constant is not needed as the boolean constant variable can be used directly

Status: Closed

C.Contract - RiskModel

High severity issues

C.1 User can withdraw more than deposited amount

Line	Code/Function

46	function userWithdraw(address to, uint256 amount) external onlyOwner {
	require(amount > 0, "Amount must be greater than 0");
	require(to != self, "Cannot withdraw to self");
	require(getBalance() >= amount, "Insufficient balance");
	
	ERC20(RITE).safeTransfer(to, amount);
	
	emit Withdrawn(to, amount);
	}

Description

There must be a mapping to keep track of how much a user deposited in the Vault. Currently, it is possible that a user can withdraw more than the amount he deposited. Although it will be the owner at the end who will be calling this withdraw function to withdraw these tokens to the user, the owner itself can make mistakes.

Remediation

Add a mapping to keep track of the amount deposited by every user and allow only that much amount of tokens to be withdrawn via userWithdraw() function.

Status: Closed

Medium severity issues

C.2 userWithdraw() function has centralization risk

Line	Code/Function

59	for (uint256 i = 0; i < count; i++) {
	address beneficiary = _vestingDetails[i].beneficiary;
	//Check if beneficiary already has a vesting

Description
userWithdraw() function is an onlyOwner function. This means that a user will need to rely on the owner in order to withdraw his/her tokens. This introduces centralization risk. A compromised or malicious owner can deny the withdrawal of a user's funds for a long time or he can also withdraw to himself instead.

Remediation
Consider using a push and pull model in which the first step (push) is where a user's funds become available to withdraw when certain conditions are met. And the second step (pull) is where the users can themselves claim the available tokens and withdraw it to their own wallet. This also removes the reliance on the owner and removes centralization risks.

Status: Acknowledged
Comment: The client said that it was designed this way, as they wanted the user to deposit token into the vault contract without paying any gas fee. They had setAllowanceWithSignature function in the token contract , once user approves the amount of token, then owner calls userDeposit() with user's address and same approved amount. In this way users don't have to pay any of the gas fee.

Low severity issues

C.3 Missing zero address validation

Line	Code/Function

16	constructor(address _RITE) {
	self = address(this);
	RITE = _RITE;
	}

34	function userDeposit(address from, uint256 amount) external onlyOwner
	{
	require(amount > 0, "Amount must be greater than 0");
	require(from != self, "Cannot deposit from self");
	
	ERC20(RITE).safeTransferFrom(from, self, amount);
	
	emit Deposited(from, amount);
	}

Line	Code/Function

46	function userWithdraw(address to, uint256 amount) external onlyOwner {
	require(amount > 0, "Amount must be greater than 0");
	require(to != self, "Cannot withdraw to self");
	require(getBalance() >= amount, "Insufficient balance");
	
	ERC20(RITE).safeTransfer(to, amount);
	
	emit Withdrawn(to, amount);
	}

Description
Missing zero address check for _RITE in constructor, from address in userDeposit() function and to address in userWithdraw() function.

Remediation
It is advised to add a require check for the same.

Status: Closed

Informational Issues

C.4 Missing event for Critical function

Line	Code/Function

16	function withdraw() external onlyOwner {
	//Balance of the vault
	uint256 amount = ERC20(RITE).balanceOf(self);
	ERC20(RITE).safeTransfer(msg.sender, amount);
	}

Description
Missing event for critical withdraw() function

Remediation
Add and emit event for withdraw() function as it is a critical operation. This would be a best practice for offchain monitoring

Status: Closed

C.5 userDeposit() need not be an onlyOwner function

Line	Code/Function

34	function userDeposit(address from, uint256 amount) external onlyOwner
	{
	require(amount > 0, "Amount must be greater than 0");
	require(from != self, "Cannot deposit from self");
	
	ERC20(RITE).safeTransferFrom(from, self, amount);
	
	emit Deposited(from, amount);
	}

Description

UserDeposit() is an onlyOwner function. According to business logic it was done to enable gasless transactions. This means if a user wants to deposit any token into the vault, he will first need to approve the contract for say x amount of tokens and then inform the owner to call the userDeposit() function in order to transfer x amount of tokens from user to the Vault. This approach is less user-friendly. It also adds a centralization risk. Also this approach is not very scalable.

Remediation

It is advised to use Biconomy or any other equivalent tool/protocol to enable gasless transactions and remove the need for userDeposit function to be onlyOwner.

Status: Acknowledged
Comment: Same as that of C.2

D. Common issues for SaleVesting, TeamVesting and Vault

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

Informational Issues

1. Renounce Ownership

   Description
   Usually, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities on his behalf. The renounceOwnership function is used in smart contracts to renounce ownership. Otherwise, if the contract's ownership has not been transferred previously, it will never have an Owner, which is risky.

   Remediation
   It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.
   Refer this post for additional info-
   https://www.linkedin.com/posts/razzor_github-razzorsecrazzorsec-contracts-activity-6873251560864968705-HOS8

   Status: Closed
   Comment: The client did overriding the renounceownership() function to transfer the ownership to a hardcoded address in the smart contract.

2. Remove pragma abicoder v2 since it is redundant. 0.8.0 compiler versions and above use pragma abicoder v2 by default.

   Status: Closed

3. It is advised to not use the latest compiler version as it can contain undiscovered bugs. Consider using version 0.8.4
   Refer- https://secureum.substack.com/p/security-pitfalls-and-best-practices-101?s=r

   Status: Closed

4. **Unlocked pragma:** Contracts should be deployed using the same compiler version/flags with which they have been tested. Locking the pragma for e.g. by not using ^ in pragma solidity 0.5.10) ensures that contracts do not accidentally get deployed using an older compiler version with unfixed bugs.

   Status: Closed

Functional Tests

Complete functional testing report has been attached below:
ritestream

Some of the tests performed are mentioned below:

SaleVesting

- ☑ Should be able to setVesting
- ☑ Should be able to set TGEDate at deployment
- ☑ Should be able to setTGEDate
- ☑ Should be able to transferOwnership
- ☑ Should be able to claim
- ☑ Should revert if time now is less than TGEDate
- ☑ Should revert if beneficiary already has vesting going on
- ☑ Should revert if beneficiary does not have a vesting amount
- ☑ Should revert if beneficiary try to claim all of their vesting
- ☑ Should revert if claim start date is greater than TGEDate
- ☑ Should revert if _date is greater than time now when setTGEDate is called
- ☑ Should revert if beneficiary vestingAmount is not greater than Zero
- ☑ Should revert if beneficiary claim start time is less than TGEDate

TeamVesting

- ☑ Should be able to setTeamVesting
- ☑ Should be able to set TGEDate at deployment
- ☑ Should be able to setStartDate
- ☑ Should be able to transferOwnership
- ☑ Should be able to claim
- ☑ Should be able to terminateNow
- ☑ Should revert if block.timstamp is greater than claimStartTime
- ☑ Should revert if beneficiary already has vesting going on
- ☑ Should revert if beneficiary does not have a vesting amount
- ☑ Should revert if vesting duration is less than zero
- ☑ Should revert if initialAmount is less than zero
- ☑ Should revert if initalClaim is equal true

- ☑ Should revert if claimStartTime is greater than block.timstamp
- ☑ Should revert if exist is equal false
- ☑ Should revert if initalClaimed is true or initialAmount is less than zero

Vault

- ☑ Should be able userDeposit
- ☑ Should be able to withdraw to address(0)
- ☑ Should be able to userWithdraw
- ☑ Should able to transferOwnership
- ☑ Should be able withdraw
- ☑ Should be able to renounceOwnership
- ☑ Should revert if address to is equal to self address when userWithdraw is called
- ☑ Should revert if amount is equal to zero when userWithdraw is called

Automated Tests

Slither

```
Token.getMessageHash(address,address,uint256).owner (contracts/Token.sol#69) shadows:
        - Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#35-37) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

SaleVesting.setTGEDate(uint256) (contracts/SaleVesting.sol#198-201) should emit an event for:
        - TGEDate = _date (contracts/SaleVesting.sol#200)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

SaleVesting.constructor(address,uint256)._RITE (contracts/SaleVesting.sol#36) lacks a zero-check on :
                - RITE = _RITE (contracts/SaleVesting.sol#38)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in SaleVesting.claim() (contracts/SaleVesting.sol#122-181):
        External calls:
        - ERC20(RITE).safeTransfer(beneficiary,amountToClaim) (contracts/SaleVesting.sol#178)
        Event emitted after the call(s):
        - Claimed(beneficiary,amountToClaim) (contracts/SaleVesting.sol#180)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

SaleVesting.setVesting(VestingDetail[]) (contracts/SaleVesting.sol#48-117) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp < TGEDate,TGE already finished, no more vesting) (contracts/SaleVesting.sol#55-58)
SaleVesting.claim() (contracts/SaleVesting.sol#122-181) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(TGEDate > 0 && block.timestamp >= TGEDate,Claim is not allowed before TGE start) (contracts/SaleVesting.sol#125-128)
        - block.timestamp >= vestingDetails[beneficiary].claimStartTime (contracts/SaleVesting.sol#153)
        - amountToClaim > vestingDetails[beneficiary].vestingAmount - vestingDetails[beneficiary].claimedAmount (contracts/SaleVesting.sol#167-169)
SaleVesting.setTGEDate(uint256) (contracts/SaleVesting.sol#198-201) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_date > block.timestamp,TGE date is not valid) (contracts/SaleVesting.sol#199)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#201-221) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#213-216)
Token.splitSignature(bytes) (contracts/Token.sol#116-141) uses assembly
        - INLINE ASM (contracts/Token.sol#131-138)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

SaleVesting.setVesting(VestingDetail[]) (contracts/SaleVesting.sol#48-117) compares to a boolean constant:
        -require(bool,string)(_vestingDetails[i].initialClaimed == false,Initial claimed is not valid) (contracts/SaleVesting.sol#94-97)
SaleVesting.claim() (contracts/SaleVesting.sol#122-181) compares to a boolean constant:
        -vestingDetails[beneficiary].initialClaimed == false && vestingDetails[beneficiary].initialAmount > 0 (contracts/SaleVesting.sol#145-146)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Different versions of Solidity is used:
        - Version used: ['0.8.9', '^0.8.0', '^0.8.1']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - 0.8.9 (contracts/SaleVesting.sol#1)
        - v2 (contracts/SaleVesting.sol#2)
        - 0.8.9 (contracts/Token.sol#2)
        - v2 (contracts/Token.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#85-87) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#114-120) is never used and should be removed
Address.functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#174-176) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193) is never used and should be removed
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#147-149) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166) is never used and should be removed
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65) is never used and should be removed
Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#21-23) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#45-58) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#69-80) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#60-67) is never used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#29-36) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12
/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (contracts/SaleVesting.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (contracts/Token.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#63)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#128-139):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#164)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter SaleVesting.setVesting(VestingDetail[])._vestingDetails (contracts/SaleVesting.sol#48) is not in mixedCase
Parameter SaleVesting.getBeneficiaryVesting(address)._beneficiary (contracts/SaleVesting.sol#186) is not in mixedCase
Parameter SaleVesting.setTGEDate(uint256)._date (contracts/SaleVesting.sol#198) is not in mixedCase
Variable SaleVesting.RITE (contracts/SaleVesting.sol#33) is not in mixedCase
Variable SaleVesting.TGEDate (contracts/SaleVesting.sol#34) is not in mixedCase
Constant Token.fixedOwnerAddress (contracts/Token.sol#13-14) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

renounceOwnership() should be declared external:
```

```
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12
/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (contracts/SaleVesting.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (contracts/Token.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#63)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#128-139):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#164)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter SaleVesting.setVesting(VestingDetail[])._vestingDetails (contracts/SaleVesting.sol#48) is not in mixedCase
Parameter SaleVesting.getBeneficiaryVesting(address)._beneficiary (contracts/SaleVesting.sol#186) is not in mixedCase
Parameter SaleVesting.setTGEDate(uint256)._date (contracts/SaleVesting.sol#198) is not in mixedCase
Variable SaleVesting.RITE (contracts/SaleVesting.sol#33) is not in mixedCase
Variable SaleVesting.TGEDate (contracts/SaleVesting.sol#34) is not in mixedCase
Constant Token.fixedOwnerAddress (contracts/Token.sol#13-14) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#54-56)
        - Token.renounceOwnership() (contracts/Token.sol#144-146)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#62-65)
name() should be declared external:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#62-64)
symbol() should be declared external:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#70-72)
decimals() should be declared external:
        - ERC20.decimals() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#87-89)
        - Token.decimals() (contracts/Token.sol#33-35)
totalSupply() should be declared external:
        - ERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#94-96)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#101-103)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#113-117)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#136-140)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#158-167)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#181-185)
decreaseAllowance(address,uint256) should be declared external:
```

```
Token.getMessageHash(address,address,uint256).owner (contracts/Token.sol#69) shadows:
        - Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#35-37) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

TeamVesting.setStartDate(uint256) (contracts/TeamVesting.sol#215-218) should emit an event for:
        - startDate = _startDate (contracts/TeamVesting.sol#217)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

TeamVesting.constructor(address,uint256)._RITE (contracts/TeamVesting.sol#38) lacks a zero-check on :
                - RITE = _RITE (contracts/TeamVesting.sol#40)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in TeamVesting.claim() (contracts/TeamVesting.sol#122-182):
        External calls:
        - ERC20(RITE).safeTransfer(beneficiary,amountToClaim) (contracts/TeamVesting.sol#179)
        Event emitted after the call(s):
        - Claimed(beneficiary,amountToClaim) (contracts/TeamVesting.sol#181)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

TeamVesting.setTeamVesting(VestingDetail[]) (contracts/TeamVesting.sol#50-120) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_vestingDetails[i].claimStartTime > block.timestamp,Beneficiary has no claimStartTime) (contracts/TeamVesting.sol#81-84)
TeamVesting.claim() (contracts/TeamVesting.sol#122-182) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(startDate != 0 && block.timestamp > startDate,Vesting period has not started) (contracts/TeamVesting.sol#123-126)
        - require(bool,string)(block.timestamp > vestingDetails[beneficiary].claimStartTime,Claiming period has not started) (contracts/TeamVesting.sol#136-139)
        - amountToClaim > vestingDetails[beneficiary].vestingAmount - vestingDetails[beneficiary].claimedAmount (contracts/TeamVesting.sol#169-171)
TeamVesting.setStartDate(uint256) (contracts/TeamVesting.sol#215-218) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_startDate > block.timestamp,Start date is in the past) (contracts/TeamVesting.sol#216)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#201-221) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#213-216)
Token.splitSignature(bytes) (contracts/Token.sol#116-141) uses assembly
        - INLINE ASM (contracts/Token.sol#131-138)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

TeamVesting.setTeamVesting(VestingDetail[]) (contracts/TeamVesting.sol#50-120) compares to a boolean constant:
        -require(bool,string)(_vestingDetails[i].initialClaimed == false,Initial claimed can not be true) (contracts/TeamVesting.sol#101-104)
TeamVesting.claim() (contracts/TeamVesting.sol#122-182) compares to a boolean constant:
        -require(bool,string)(vestingDetails[beneficiary].exists == true,Beneficiary has terminated) (contracts/TeamVesting.sol#132-135)
TeamVesting.claim() (contracts/TeamVesting.sol#122-182) compares to a boolean constant:
        -vestingDetails[beneficiary].initialClaimed == false && vestingDetails[beneficiary].initialAmount > 0 (contracts/TeamVesting.sol#151-152)
TeamVesting.terminateNow(address) (contracts/TeamVesting.sol#197-211) compares to a boolean constant:
        -require(bool,string)(vestingDetails[beneficiary].exists == true,Beneficiary is already terminated) (contracts/TeamVesting.sol#204-207)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Different versions of Solidity is used:
        - Version used: ['0.8.9', '^0.8.0', '^0.8.1']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
```

```
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - 0.8.9 (contracts/TeamVesting.sol#1)
        - v2 (contracts/TeamVesting.sol#2)
        - 0.8.9 (contracts/Token.sol#2)
        - v2 (contracts/Token.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#85-87) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#114-120) is never used and should be removed
Address.functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#174-176) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193) is never used and should be removed
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#147-149) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166) is never used and should be removed
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65) is never used and should be removed
Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#21-23) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#45-58) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#69-80) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#60-67) is never used and should be removed
SafeERC20.safeTransferFrom(IERC20,address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#29-36) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12
/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (contracts/TeamVesting.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (contracts/Token.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#63)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#128-139):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#164)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter TeamVesting.setTeamVesting(VestingDetail[])._vestingDetails (contracts/TeamVesting.sol#50) is not in mixedCase
Parameter TeamVesting.setStartDate(uint256)._startDate (contracts/TeamVesting.sol#215) is not in mixedCase
Variable TeamVesting.RITE (contracts/TeamVesting.sol#35) is not in mixedCase
Constant Token.fixedOwnerAddress (contracts/Token.sol#13-14) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
Token.getMessageHash(address,address,uint256).owner (contracts/Token.sol#69) shadows:
        - Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#35-37) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Vault.constructor(address)._RITE (contracts/Vault.sol#16) lacks a zero-check on :
                - RITE = _RITE (contracts/Vault.sol#18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in Vault.userDeposit(address,uint256) (contracts/Vault.sol#34-41):
        External calls:
        - ERC20(RITE).safeTransferFrom(from,self,amount) (contracts/Vault.sol#38)
        Event emitted after the call(s):
        - Deposited(from,amount) (contracts/Vault.sol#40)
Reentrancy in Vault.userWithdraw(address,uint256) (contracts/Vault.sol#46-54):
        External calls:
        - ERC20(RITE).safeTransfer(to,amount) (contracts/Vault.sol#51)
        Event emitted after the call(s):
        - Withdrawn(to,amount) (contracts/Vault.sol#53)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Address.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#201-221) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#213-216)
Token.splitSignature(bytes) (contracts/Token.sol#116-141) uses assembly
        - INLINE ASM (contracts/Token.sol#131-138)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity is used:
        - Version used: ['0.8.9', '^0.8.0', '^0.8.1']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - 0.8.9 (contracts/Token.sol#2)
        - v2 (contracts/Token.sol#3)
        - 0.8.9 (contracts/Vault.sol#1)
        - v2 (contracts/Vault.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#85-87) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#114-120) is never used and should be removed
Address.functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#174-176) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193) is never used and should be removed
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#147-149) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166) is never used and should be removed
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65) is never used and should be removed
Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#21-23) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#45-58) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#69-80) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#60-67) is never used and should be removed
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12
/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (contracts/Token.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version0.8.9 (contracts/Vault.sol#1) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#63)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#128-139):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#137)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#157-166):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#164)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#184-193):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Constant Token.fixedOwnerAddress (contracts/Token.sol#13-14) is not in UPPER_CASE_WITH_UNDERSCORES
Variable Vault.RITE (contracts/Vault.sol#14) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#54-56)
        - Token.renounceOwnership() (contracts/Token.sol#144-146)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#62-65)
name() should be declared external:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#62-64)
symbol() should be declared external:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#70-72)
decimals() should be declared external:
        - ERC20.decimals() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#87-89)
        - Token.decimals() (contracts/Token.sol#33-35)
totalSupply() should be declared external:
        - ERC20.totalSupply() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#94-96)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#101-103)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#113-117)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#136-140)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#158-167)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#181-185)
decreaseAllowance(address,uint256) should be declared external:
```

Some issues such as missing zero address check and boolean constants were detected which were mentioned in the report. Some false positives were reported as which were not actual issues.

Mythril

No issues were reported by Mythril.

Closing Summary

Numerous issues of high,medium and low severity were discovered during the initial audit. Most of the Issues were fixed by the Auditee

Disclaimer

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

Audit Report
March, 2022

For

ritestream

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com