QuillAudits

Audit Report
July, 2023

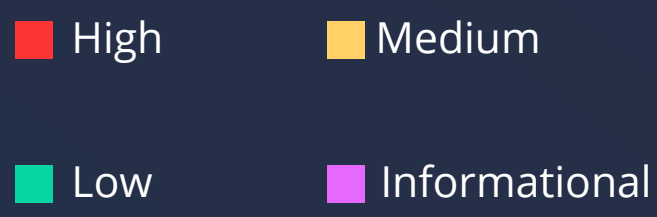
For

ritestream

Table of Content

Executive Summary

Project Name	Ritestream
Overview	Subscription contract is to facilitate the acceptance of RITE tokens as payment for subscribing to the ritestream app.
Timeline	July 12, 2023 to July 17, 2023
Audit Scope	The scope of this audit was to analyze Ritestram's Subscription smart contract for quality, security, and correctness. *https://github.com/ritestream/ritestream-contract/blob/master/contracts/Subscription.sol* Commit hash: 7522de838360160311cc0715d60d930dc7fbb78d
Fixed In	*https://github.com/ritestream/ritestream-contract/commit/51806a461f7b97b1e891cc8b316a36a248064b57#diff-dc0be7317395f63c23fc44721bd367e07f2cd2a9e410b6a9a5b494aff799b5e1* *https://github.com/ritestream/ritestream-contract/commit/83c2e25be8fedb0ac7fef242e0b5689022a56d56#diff-dc0be7317395f63c23fc44721bd367e07f2cd2a9e410b6a9a5b494aff799b5e1*



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	1	2	5

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 - Subscription.sol

High Severity Issues

No issues found

Medium Severity Issues

A.1 Subscription can be overwritten

Description

As the function signature is the same and the hash created also follows the same pattern, it is possible to overwrite a subscription by mistake.

Remediation

To fix this, it is recommended to check the existing subscription for a msg.sender and update the values like in update subscription or revert. It is also recommended to have a special identifier like a string or enum in signature to identify which function call was intended.

Status

Resolved

Low Severity Issues

A.2 Signature can be reused multiple times

Description

As the message hash created follows the same pattern in subscribe and updateSubscriptionPlan, it is possible to reuse it any number of times and increase the duration. Users will still have to pay but this can go against operator's permission unless the operator is updated.

Remediation

To fix this, it is recommended to have a nonce in the signature, which should be expired after a function call.

Status

Resolved

A.3 Check Effect Interaction pattern not followed

Description

The functions subscribe and updateSubscriptionPlan transfers the RITE tokens from msg.sender before updating contract states. As RITE token is not a part of this contract, it is assumed that it can be used to reenter the functions and exploit the function to double spend subscription in updateSubscriptionPlan.

Remediation

To fix this, it is recommended to first update the storage for a subscription and then call `transferFrom`.

Status

Resolved

Informational Issues

A.4 Missing checks

Description

It is recommended to add a check for address(0) while setting the operator as it is used in validating signatures in subscriptions.

Status

Resolved

A.5 Contracts without storage gap.

Description

It is recommended to add a gap in the contract to avoid storage collision in future if the current contract is upgraded and extended.

Status

Resolved

A.6 Missing event emission in setOperator

Description

The function setOperator does not emit an event on updating operator.

Remediation

As it is a critical state in this contract, it is recommended to emit an event as it is easy to track on blockchain.

Status

Resolved

A.7 Unwanted require statements

Description

Following require statements can be removed from line number 68 and 101 as msg.sender can never be zero address.

```
    require(msg.sender != address(0), "From address cannot be zero");
```

Status

Resolved

A.6 Missing event emission in setOperator

Description

Parameter Subscription.initialize(address)._RITE (contracts/Subscription.sol#37) is not in mixedCase
Parameter Subscription.subscribe(uint256,Signature)._signature (contracts/Subscription.sol#65) is not in mixedCase
Parameter Subscription.updateSubscriptionPlan(uint256,Signature)._signature (contracts/Subscription.sol#97) is not in mixedCase
Variable Subscription.RITE (contracts/Subscription.sol#29) is not in mixedCase
Variable Subscription._operator (contracts/Subscription.sol#30) is not in mixedCase
Constant Subscription.fixedOwnerAddress (contracts/Subscription.sol#32-33) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Token.fixedOwnerAddress (contracts/Token.sol#12-13) is not in UPPER_CASE_WITH_UNDERSCORES

Reference

https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Status

Resolved

Functional Testing

Some of the tests performed are mentioned below:

- Should get rite token address and balance of rite token after vesting contract deployed
- Should only allow owner to set operator
- Should not allow to update plan if no subscription found
- Should allow user to subscribe
- Should only allow owner to withdraw
- Should allow user to update subscription plan
- Should not overwrite subscription on calling function again with different amount.

Automated Tests

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

Subscription (contracts/Subscription.sol#25-167) is an upgradeable contract that does not protect its initialize functions: Subscription.initialize(address) (contracts/Subscription.sol#37-44). Anyone can delete the contract with: UUPSUpgradeable.upgradeTo(address) (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#74-77)UUPSUpgradeable.upgradeToAndCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#89-92)Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unprotected-upgradeable-contract

ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool).slot (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#84) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC1967UpgradeUpgradeable._upgradeToAndCall(address,bytes,bool) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#65-70) ignores return value by AddressUpgradeable.functionDelegateCall(newImplementation,data) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#68)
ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#77-91) ignores return value by IERC1822ProxiableUpgradeable(newImplementation).proxiableUUID() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#84-88)
ERC1967UpgradeUpgradeable._upgradeBeaconToAndCall(address,bytes,bool) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#156-162) ignores return value by AddressUpgradeable.functionDelegateCall(IBeaconUpgradeable(newBeacon).implementation(),data) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Token.getMessageHash(address,address,uint256).owner (contracts/Token.sol#68) shadows:
- Ownable.owner() (node_modules/@openzeppelin/contracts/access/Ownable.sol#43-45) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Subscription.setOperator(address).operator (contracts/Subscription.sol#145) lacks a zero-check on :
- _operator = operator (contracts/Subscription.sol#146)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Variable 'ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool).slot (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#84)' in ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#77-91) potentially used before declaration: require(bool,string)(slot == _IMPLEMENTATION_SLOT,ERC1967Upgrade:

unsupported proxiableUUID) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#85)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in Subscription.subscribe(uint256,Signature) (contracts/Subscription.sol#65-90):
External calls:
- ERC20(RITE).safeTransferFrom(msg.sender,self,amount) (contracts/Subscription.sol#76)
State variables written after the call(s):
- subscriptionPlans[msg.sender] = SubscriptionPlan(msg.sender,amount,block.timestamp,block.timestamp + 2592000) (contracts/Subscription.sol#78-83)
Reentrancy in Subscription.updateSubscriptionPlan(uint256,Signature) (contracts/Subscription.sol#95-129):
External calls:
- ERC20(RITE).safeTransferFrom(msg.sender,self,amount) (contracts/Subscription.sol#109)
State variables written after the call(s):
- subscriptionPlan.amountPaid += amount (contracts/Subscription.sol#120)
- subscriptionPlan.endDate += 2592000 (contracts/Subscription.sol#121)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in Subscription.subscribe(uint256,Signature) (contracts/Subscription.sol#65-90):
External calls:
- ERC20(RITE).safeTransferFrom(msg.sender,self,amount) (contracts/Subscription.sol#76)
Event emitted after the call(s):
- Subscribed(msg.sender,block.timestamp,block.timestamp + 2592000,amount) (contracts/Subscription.sol#84-89)
Reentrancy in Subscription.updateSubscriptionPlan(uint256,Signature) (contracts/Subscription.sol#95-129):
External calls:
- ERC20(RITE).safeTransferFrom(msg.sender,self,amount) (contracts/Subscription.sol#109)
Event emitted after the call(s):
- Subscribed(msg.sender,block.timestamp,block.timestamp + 2592000,amount) (contracts/Subscription.sol#123-128)
Reentrancy in Subscription.withdraw() (contracts/Subscription.sol#137-142):
External calls:
- ERC20(RITE).safeTransfer(msg.sender,amount) (contracts/Subscription.sol#140)
Event emitted after the call(s):
- Withdrawn(msg.sender,amount) (contracts/Subscription.sol#141)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

AddressUpgradeable._revert(bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#231-243) uses assembly
- INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#236-239)
StorageSlotUpgradeable.getAddressSlot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#62-67) uses assembly

- INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#64-66)
StorageSlotUpgradeable.getBooleanSlot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#72-77) uses assembly
- INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#74-76)
StorageSlotUpgradeable.getBytes32Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#82-87) uses assembly
- INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#84-86)
StorageSlotUpgradeable.getUint256Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#92-97) uses assembly
- INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#94-96)
StorageSlotUpgradeable.getStringSlot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#102-107) uses assembly
- INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#104-106)
StorageSlotUpgradeable.getStringSlot(string) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#112-117) uses assembly
- INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#114-116)
StorageSlotUpgradeable.getBytesSlot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#122-127) uses assembly
- INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#124-126)
StorageSlotUpgradeable.getBytesSlot(bytes) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#132-137) uses assembly
- INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#134-136)
Address._revert(bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#231-243) uses assembly
- INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#236-239)
Token.splitSignature(bytes) (contracts/Token.sol#115-140) uses assembly
- INLINE ASM (contracts/Token.sol#130-137)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
- Version used: ['0.8.16', '^0.8.0', '^0.8.1', '^0.8.2']
- 0.8.16 (contracts/Subscription.sol#2)
- 0.8.16 (contracts/Token.sol#2)
- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#4)
- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/IERC1967Upgradeable.sol#4)

- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/draft-IERC1822Upgradeable.sol#4)
- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/beacon/IBeaconUpgradeable.sol#4)
- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#4)
- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4)
- ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#5)
- ^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/extensions/IERC20Permit.sol#4)
- ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4)
- ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
- ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4)
- ^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4)
- ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#4)
- ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/IERC1967Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/draft-IERC1822Upgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/beacon/IBeaconUpgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#5) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4) allows old versions

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Permit.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#64-69):
- (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#67)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#128-137):
- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#155-162):
- (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#160)
Low level call in AddressUpgradeable.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#180-187):
- (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#185)
Low level call in SafeERC20._callOptionalReturnBool(IERC20,bytes) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#134-142):
- (success,returndata) = address(token).call(data) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#139)
Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#64-69):
- (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#67)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#128-137):
- (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#155-162):
- (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#160)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#180-187):

- (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#185)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function OwnableUpgradeable.__Ownable_init() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#29-31) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#33-35) is not in mixedCase
Variable OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#94) is not in mixedCase
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#20-21) is not in mixedCase
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#23-24) is not in mixedCase
Variable ERC1967UpgradeUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#169) is not in mixedCase
Function UUPSUpgradeable.__UUPSUpgradeable_init() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#23-24) is not in mixedCase
Function UUPSUpgradeable.__UUPSUpgradeable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#26-27) is not in mixedCase
Variable UUPSUpgradeable.__self (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#29) is not in mixedCase
Variable UUPSUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#111) is not in mixedCase
Function ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is not in mixedCase
Variable ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#36) is not in mixedCase
Function IERC20Permit.DOMAIN_SEPARATOR() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Permit.sol#59) is not in mixedCase
Parameter Subscription.initialize(address)._RITE (contracts/Subscription.sol#37) is not in mixedCase
Parameter Subscription.subscribe(uint256,Signature)._signature (contracts/Subscription.sol#65) is not in mixedCase
Parameter Subscription.updateSubscriptionPlan(uint256,Signature)._signature (contracts/Subscription.sol#97) is not in mixedCase
Variable Subscription.RITE (contracts/Subscription.sol#29) is not in mixedCase
Variable Subscription._operator (contracts/Subscription.sol#30) is not in mixedCase
Constant Subscription.fixedOwnerAddress (contracts/Subscription.sol#32-33) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Token.fixedOwnerAddress (contracts/Token.sol#12-13) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions. analyzed (20 contracts with 84 detectors), 73 result(s) found

Summary

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

Some issues of High, Medium, 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 Ritestream Platform. This audit does not provide a security or correctness guarantee for 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 multi-step 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.

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.

850+
Audits Completed

$30B
Secured

800K
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2023

For

ritestream

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com