HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Rand Network
Date: March 29th, 2022

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed — upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for Rand Network.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type of Contracts	Vault; SwapCurve; ERC1155 token; Stable Token; Yield Aggregator
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://www.rand.network
Timeline	23.02.2022 - 29.03.2022
Changelog	21.03.2022 - Initial Review 29.03.2022 - Revise

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Rand Network (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Repository:
https://github.com/Rand-Network/contracts

Commit:
e8a930315e93406cae9ae708398fb3309b4a9283

Technical Documentation: Yes

- https://rand-network.gitbook.io/rand-network/protocol/tokenomics
- https://drive.google.com/file/d/1ACE5Sa4SJdbyUQ8gorHaTi43_TL0C5gZ/view
- Whitepaper (in spanish)

JS tests: Yes (included in `test` folder)

Contracts:

swap/SafeMath.sol
reserve/ReserveInterface.sol
reserve/Reserve.sol
strategy/UniformRandomNumber.sol
yield/IEthAnchorRouter.sol
yield/YieldSource.sol
random/RandomNumberGenerator.sol
random/RandomNumberGeneratorInterface.sol
swap/IUniswapRouterV2.sol
test/StableERC20.sol
swap/UniswapV2Library.sol
tokens/MultiToken.sol
vaultreserve/VaultReserve.sol
tokens/StableTokenInterface.sol
yield/IYieldAggregator.sol
yield/IOperation.sol
tokens/UsdcTokenInterface.sol
tokens/UsdcToken.sol
party/PartiesStorageLibrary.sol
vaultmanager/VaultsStorageLibrary.sol
vault/VaultInterface.sol
swap/ICurve.sol
yield/YieldAggregator.sol
vault/Vault.sol
swap/SwapCurve.sol
test/RandomNumberGeneratorMock.sol
party/PartyManager.sol
test/LinkMock.sol
tokens/YieldAggregatorLP.sol
vaultmanager/VaultManager.sol
swap/ISwapper.sol
swap/UniswapRouter.sol
strategy/PrizeStrategy.sol

```
        vaultmanager/VaultManagerInterface.sol
        strategy/PrizeStrategyInterface.sol
        tokens/StableToken.sol
        test/VrfCoordinatorMock.sol
```

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

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ EIP standards violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency ▪ Kill-Switch Mechanism

Executive Summary

The score measurements details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided whitepaper, smart-contract architecture, tokenomics, and some technical requirements. Total Documentation Quality score is **8** out of **10**.

Code quality

The total CodeQuality score is **6** out of **10**. Unit tests were provided, but some of them are failing. Lots of NetSpecs. Mixed solidity versions.

Architecture quality

The architecture quality score is **9** out of **10**. All the logic is split into logical parts. Pretty readable and understandable. No detailed architecture description.

Security score

As a result of the audit, security engineers found **1** medium severity issue. The security score is **10** out of **10**. All found issues are displayed in the "Issues overview" section.

Summary

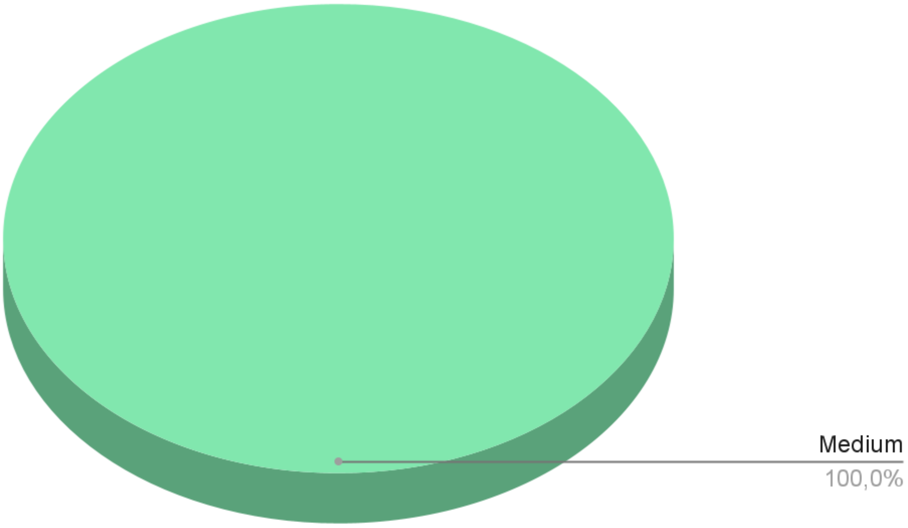
According to the assessment, the Customer's smart contract has the following score: **9.3**

1
2
3
4
5
6
7
8
9
10
You are here

Notices

1. BACKEND_ADMIN_ROLE can withdraw any amount of USDC tokens from the VaultReserve contract.

Graph 1. The distribution of vulnerabilities after the audit.



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

AS-IS overview

RandomNumberGenerator

Description

Contract in charge of storing and managing the random number requests provided by the Chainlink Defender

- `constructor(address _vrfCoordinator, address _link)` (public)

Public constructor

- `getLink() → address` (external)

Returns the address of LINK Token used by Random Number Generator

- `setKeyHash(bytes32 _keyhash)` (external)

Allows owner to set the Chainlink VRF keyhash

- `setFee(uint256 _fee)` (external)

Allows owner to set the Fee per request required by the Chainlink VRF

- `setWithdrawOffset(uint256 _offset)` (external)

Allows owner and backend to set offset to check near zero

- `getRequestFee() → address feeToken, uint256 requestFee` (external)

Gets the Fee for making a Request against an RNG service

- `requestRandomNumber(uint256 _vaultId) → uint32 lockBlock` (external)

Sends a request for a random number to the 3rd-vault service

Some services will complete the request immediately, others may have a time-delay

Some services require payment in the form of a token, such as $LINK for Chainlink VRF

- `isRequestComplete(uint256 _vaultId) → bool isCompleted` (external)

Checks if the request for randomness from the 3rd-vault service has been completed

for time-delayed requests, this function is used to check/confirm completion

- `getLockBlock(uint256 _vaultId) → uint256 lockBlock` (external)

Returns the block number at which the RNG service will start generating time-delayed randomness

- `getRandomNumber(uint256 _vaultId) → uint256 randomNumber` (external)

Gets the random number produced by the 3rd-vault service

- `setVaultManagerAddress(address _vaultManagerAddress)` (external)

Used to change the address of the Vault Manager contract

- `_requestRandomness() → bytes32 requestId` (internal)

Requests a new random number from the Chainlink VRF

The result of the request is returned in the function `fulfillRandomness`

- `fulfillRandomness(bytes32 requestId, uint256 randomness)` (internal)

Callback function used by VRF Coordinator

The VRF Coordinator will only send this function verified responses.

The VRF Coordinator will not pass randomness that could not be verified.

- `withdrawLink(address _account, uint256 _amount_link)` (external)

Used by Admin to withdraw LINK tokens

PrizeStrategy

Description

Contract responsible for calculating users' contributions and sharing their rewards

- `initialize(address _multiTokenAddress)` (external)

Initialize init the contract with the following parameters

This function is called only once during the contract initialization

- `setPrizeParameters(uint256 _vaultId, uint256[] _tiers, uint256[] _percentages, uint256 _prize, uint256 _rewardCoefficient)` (external)

Used to set up the parameters necessary for calculating the prizes for a vault

- `computeRewards(uint256 _vaultId, address[] _users)` (external)

Used to reward and increase the chance of winning for users, which kept a part of their balance in the vault since the last prize distribution

- `getWinners(uint256 _vaultId, address[] _users, uint256 _totalBalance, uint256 _randomNumber) → address[] winners, uint256[] winnersTiers` (external)

Used to calculate the winner's address and the tier that they won after the vault has finished

- `getWinnersPrizes(uint256 _vaultId, uint256[] _tiers) → uint256[] winnersPrizes` (external)

Used to calculate the winners prizes based on the tire that they have won

- `setVaultManagerAddress(address _vaultManagerAddress)` (external)

Used to change the address of the Vault Manager contract

- `setMultiTokenAddress(address _multiTokenAddress)` (external)

Used to change the address of the Rand Multi Token contract

- `_computeUserReward(uint256 _vaultId, address _user)` (internal)

Used to reward and increase the chance of winning for a participant, which kept a part of his balance in the vault since the last prize distribution

- `_getTierRandomNumbers(uint256 _randomNumber, uint256[] _tierPrizes, uint256 _totalBalance) → struct PrizeStrategy.winnerRandomNumber[] randomNumbersBuffer` (internal)

Used to get for each prize a pseudo-random number

- `_sort(struct PrizeStrategy.winnerRandomNumber[] data) → struct PrizeStrategy.winnerRandomNumber[]` (public)

Used to sort the pseudo-random numbers array

- `_quickSort(struct PrizeStrategy.winnerRandomNumber[] arr, int256 left, int256 right)` (internal) winnerRandomNumber

- □ uint256 randomNumber
- □ uint256 tier

indexesHolder

- □ uint256 currentIndex
- □ uint256 currentUserIndex
- □ uint256 winnersIndex
- □ uint256 bufferIndex

UniformRandomNumber

Description

A library that uses entropy to select a random number within a bound. Compensates for modulo bias.

Thanks to
https://medium.com/hownetworks/dont-waste-cycles-with-modulo-bias-35b6fdafcf94

- `uniform(uint256 _entropy, uint256 _upperBound) → uint256` (internal)

Select a random number without modulo bias using a random seed and upper bound

SwapCurve

- `initialize(address _ustPoolSwapAddress, address _ustTokenAddress, address _usdcTokenAddress, address _backendAddress)` (external)

Initialize init the contract with the following parameters

This function is called only once during the contract initialization

- `setRoutes()` (internal)
- `swapToken(address _from, address _to, uint256 _amount, uint256 _minAmountOut, address _beneficiary)` (public)

Swaps two tokens (USDC and wUST) using Curve.fi

- `setUstPoolSwapAddress(address _ustPoolSwapAddress)` (external)

Used to change the address of the Curve's UST Pool Swap Address

- `setBackendAddress(address _backendAddress)` (external)

Used to change the address of the Vault contract

Route

- □ int128 _fromIx
- □ int128 _toIx

RandMultiToken

Description

ERC1155 contract where each token represents a vault and the balances of the token represents the balances of the user for the vault

Each token has a normal balance and a time-weighted one. The time-weighted balance represents the contribution of the user to the vault

The fees for late deposits or early withdraws are reflected in the time-weighted balance

- `initialize()` (external)

Initialize init the contract with the following parameters

This function is called only once during the contract initialization

- `__ERC1155_init(string uri_)` (internal)

See {_setURI}.

- `__ERC1155_init_unchained(string uri_)` (internal)
- `supportsInterface(bytes4 interfaceId) → bool` (public)

See {IERC165-supportsInterface}.

- `uri(uint256) → string` (public)

See {IERC1155MetadataURI-uri}.

This implementation returns the same URI for *all* token types. It relies on the token type ID substitution mechanism

https://eips.ethereum.org/EIPS/eip-1155#metadata[defined in the EIP].

Clients calling this function must replace the `\{id\}` substring with the actual token type ID.

- `mint(address account, uint256 id, uint256 amount, uint256 twaAmount, bytes data)` (public)
- `burn(address account, uint256 id, uint256 amount, uint256 twaAmount)` (public)
- `setReward(address account, uint256 id, uint256 amount)` (public)
- `setPrize(address account, uint256 id, uint256 amount)` (public)
- `balanceOf(address account, uint256 id) → uint256` (public)

See {IERC1155-balanceOf}.

Requirements:

- `account` cannot be the zero address
- `twaBalanceOf(address account, uint256 id) → uint256` (public)

See {IERC1155-balanceOf}.

Requirements:

- `account` cannot be the zero address
- `balancesOf(address account, uint256 id) → uint256 amount, uint256 twaAmount` (public)
- `balanceOfBatch(address[] accounts, uint256[] ids) → uint256[]` (public)

See {IERC1155-balanceOfBatch}.

Requirements:

- `accounts` and `ids` must have the same length
- `rewardOf(address account, uint256 id) → uint256` (public)

Used to get the vault prize won by an account, 0 in case it did not win anything

- `prizeOf(address account, uint256 id) → uint256` (public)

Used to get the vault prize won by an account, 0 in case it did not win anything

- `twaBalanceOfBatch(address[] accounts, uint256[] ids) → uint256[]` (public)

See {IERC1155-balanceOfBatch}.

Requirements:

- `accounts` and `ids` must have the same length
- `setApprovalForAll(address operator, bool approved)` (public)

See {IERC1155-setApprovalForAll}.

- `isApprovedForAll(address account, address operator) → bool` (public)

See {IERC1155-isApprovedForAll}.

- `safeTransferFrom(address from, address to, uint256 id, uint256 amount, bytes data)` (public)

See {IERC1155-safeTransferFrom}.

- `safeBatchTransferFrom(address from, address to, uint256[] ids, uint256[] amounts, bytes data)` (public)

See {IERC1155-safeBatchTransferFrom}.

- `calculateTwaAmount(uint256 id, address account, uint256 realAmount) → uint256 _twaAmount` (public)

Used to calculate the twa amount that needs to be transferred

Based on the real amount and balances of the user

- `updateParticipantLastWithdrawDate(uint256 id, address account)` (external)

Used to update the date of the last withdraw for a participant

Called inside the participantWithdraw function of the Vault Manager

- `getParticipantLastWithdrawDate(uint256 id, address account) → uint256` (external)
- `updateParticipantLastDepositDate(uint256 id, address account)` (external)

Used to update the date of the last deposit for a user

Called inside the participantDeposit function of the Vault Manager

- `getParticipantLastDepositDate(uint256 id, address account) → uint256` (external)
- `grantMintAndBurnRights(address _address)` (external)

Used to grant an address the rights to mint and burn

Rand internal token

- `grantAdminRole(address _address)` (external)

Used to grant and address admin rights

- `_safeTransferFrom(address from, address to, uint256 id, uint256 amount, bytes data)` (internal)

Transfers `amount` tokens of token type `id` from `from` to `to`.

Emits a {TransferSingle} event.

Requirements:

- `to` cannot be the zero address
- `from` must have a balance of tokens of type `id` of at least `amount`
- If `to` refers to a smart contract, it must implement {IERC1155Receiver-onERC1155Received} and return the acceptance magic value
- `_safeBatchTransferFrom(address from, address to, uint256[] ids, uint256[] amounts, bytes data)` (internal)

xref:ROOT:erc1155.adoc#batch-operations[Batched] version of {_safeTransferFrom}.

Emits a {TransferBatch} event.

Requirements:

- If `to` refers to a smart contract, it must implement {IERC1155Receiver-onERC1155BatchReceived} and return the acceptance magic value
- `_setURI(string newuri)` (internal)

Sets a new URI for all token types by relying on the token type ID

substitution mechanism

https://eips.ethereum.org/EIPS/eip-1155#metadata[defined in the EIP].

By this mechanism, any occurrence of the `\{id\}` substring in either the URI or any of the amounts in the JSON file at said URI would be replaced by

clients with the token type ID.

For example, the `https://token-cdn-domain/\{id\}.json` URI would be

interpreted by clients as

`https://token-cdn-domain/000000000000000000000000000000000000000000000000000000000004cce0.json`

for token type ID 0x4cce0.

See {uri}.

Because these URIs cannot be meaningfully represented by the {URI} event, this function emits no events.

- `_mint(address account, uint256 id, uint256 amount, uint256 twaAmount, bytes data)` (internal)

Creates `amount` tokens of token type `id` and assigns them to `account`.

Emits a {TransferSingle} event.

Requirements:

- account cannot be the zero address
- If account refers to a smart contract, it must implement {IERC1155Receiver-onERC1155Received} and return the acceptance magic value
- _mintBatch(address to, uint256[] ids, uint256[] amounts, uint256[] twaAmounts, bytes data) (internal)

xref:ROOT:erc1155.adoc#batch-operations[Batched] version of {_mint}.

Requirements:

- ids and amounts must have the same length
- If to refers to a smart contract, it must implement {IERC1155Receiver-onERC1155BatchReceived} and return the acceptance magic value
- _burn(address account, uint256 id, uint256 amount, uint256 twaAmount) (internal)

Destroys amount tokens of token type id from account

Requirements:

- account cannot be the zero address.
- account must have at least amount tokens of token type id.
- _burnBatch(address account, uint256[] ids, uint256[] amounts) (internal)

xref:ROOT:erc1155.adoc#batch-operations[Batched] version of {_burn}.

Requirements:

- ids and amounts must have the same length.
- _beforeTokenTransfer(address operator, address from, address to, uint256[] ids, uint256[] amounts, bytes data) (internal)

Hook that is called before any token transfer. This includes minting and burning, as well as batched variants.

The same hook is called on both single and batched variants. For single transfers, the length of the id and amount arrays will be 1.

Calling conditions (for each id and amount pair):

- When from and to are both non-zero, amount of from's tokens of token type id will be transferred to to.
- When from is zero, amount tokens of token type id will be minted for to.
- When to is zero, amount of from's tokens of token type id will be burned.
- from and to are never both zero.
- ids and amounts have the same, non-zero length.

To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].

RandStableToken

Description

Contract holding all the tokens associated to the USDC balance of the user.

It uses a ERC-20 implementation where each token is backed by one USDC token

The participants in Rand Network hold as many tokens as the USDC token they have deposited

- `initialize()` (external)

Initialize init the contract with the following parameters

This function is called only once during the contract initialization

- `mint(address _account, uint256 _amount)` (public)

Creates 'amount' of tokens and assigns them to 'account' balance, increasing the total supply.

- `burn(address _account, uint256 _amount)` (public)

Removes 'amount' of tokens from the 'account' balance, decreasing the total supply.

- `approveExternal(address _from, address _to, uint256 _amount)` (public)

Approves 'amount' of tokens to be transferred from the 'from' to 'to'

- `grantMintAndBurnRights(address _address)` (external)

used to grant an address the rights to mint and burn Rand internal token

VaultManager

Description

Contract in charge of storing and managing the different vaults created by the community

- `initialize(address _multiTokenAddress, address _stableERC20Address, address _backendAddress)` (external)

Initialize init the contract with the following parameters

This function is called only once during the contract initialization

- `createVault(uint256 _vaultId, address _creator, uint256 _minimumParticipants, uint256 _lockDuration, bool _isRecurrent, bool _isPublic) → uint256 vaultId` (public)

It creates a new vault with the configuration given by the creator

- `getVault(uint256 _vaultId) → struct VaultsStorageLibrary.Vault vault` (external)

It returns the attributes of a specific vault

- `updateLockDuration(uint256 _vaultId, uint256 _lockDuration)` (external)

Used to update the duration for which the participants cannot withdraw after depositing

- `getParticipants(uint256 _vaultId) → address[] participants` (external)

It returns an array containing the address of the participants to the vault

- `isParticipant(uint256 _vaultId, address _participant) → bool` (public)

It checks if an address is a selected participant for a specific vault

- `participantDeposit(uint256 _vaultId, address _participant, uint256 _amount)` (public)

It allows a valid participant to deposit some amount for joining a vault

Requirements:

- The vault needs to be in `WaitingParticipants` state

- The `amount` should be the one defined by the vault creator

- Only selected participants can join

- `participantWithdraw(uint256 _vaultId, address _participant, uint256 _amount, uint256 _feeCoefficent)` (public)

It allows backend to withdraw Rand Multi Tokens from the vault for a user in exchange of Rand Stable Tokens

- `closeVaultBeforeYielding(uint256 _vaultId)` (external)

It allows closing a particular vault before Yielding and withdrawing all funds with no fee

- `startYielding(uint256 _vaultId, uint256 _duration, bool _forceStart)` (public)

Used by the backend to mark the beginning of the yielding period by changing the vault state to yielding

- `stopYielding(uint256 _vaultId, uint256[] _tiers, uint256[] _percentages, uint256 _prize, uint256 _rewardCoefficient, uint256 _externalRandomNumber, bool _timeWeightNonRadom, bool _allToCreatorNonRandom, bool _tierAndRandomBased)` (public)

If the vault was yielding and the duration is over, it allows to stop the yielding phase and start the rewarding phase

Everybody can call this method!

- `distributeRewards(uint256 _vaultId, uint256 _randomNumber)` (external)
- `rewardDistribution(uint256 _vaultId, uint256 _randomNumber)` (internal)

Passed the vault id, and the random number is calculated

Function created to by pass the randomly generated number by chainlink

Whenever the backend passes the random number, this function could be called internally

- `transferAndBurn(uint256 _vaultId, address _participant)` (internal)

Used to transfer Rand Stable Tokens from the Vault to an account and burn the same amount of Rand Multi Tokens from the account balance

- `hashVaultId(address _owner, uint256 _minimumParticipants, uint256 _duration) → bytes32` (public)

It generates a hash that can be used as unique vaultId

- `_handleRewards(uint256 _vaultId, address[] _participants, uint256 _vaultTwaBalance, uint256 _randomNumber)` (internal)

Used to handle the distribution of the rewards and prepares the vault for the next stage

- `_distributeRewards(uint256 _vaultId, address[] _participants, uint256 _vaultTwaBalance, uint256 _randomNumber)` (internal)

Used to get the winners and prizes from the Prize Strategy and mint the tokens for the winners

- `_distBasedOnTwaNonRandom(uint256 _vaultId, uint256 _prize)` (internal)
- `_distSendAllToCreatorNonRandom(uint256 _vaultId, uint256 _prize)` (internal)
- `_handleParticipantWithdraw(uint256 _vaultId, address _participant, uint256 _amount, uint256 _feeAmount)` (internal)

Handles the funds when the participant is withdrawing

- `_burn(uint256 _amount)` (internal)

Uses RandStableTokenInterface to burn the Rand internal tokens obtained when users are joining to a vault

- `_isOpen(enum VaultsStorageLibrary.VaultState _state) → bool` (internal)

Checks if a vault is open for the users to join

- `_isWaiting(enum VaultsStorageLibrary.VaultState _state) → bool` (internal)

Checks if a vault is open for the users to join

- `_verifyPrizesLessThanParticipants(uint256 _vaultId, uint256[] _tiers) → bool` (internal)
- `setBackendAddress(address _backendAddress)` (external)

Used to change the address of the Vault contract

- `setRngAddress(address _rngAddress)` (external)

Used to change the address of the Random Number Generator contract

- `setPrizeStrategyAddress(address _prizeStrategyAddress)` (external)

Used to change the address of the Prize Strategy contract

VaultsStorageLibrary

- create(struct VaultsStorageLibrary.VaultsList _self, uint256 _vaultId, address _creator, uint256 _minimumParticipants, uint256 _lockDuration, bool _isReccurrent, bool _isPublic) → uint256 (external)
- addParticipant(struct VaultsStorageLibrary.VaultsList _self, uint256 _vaultId, address _participant) (external)
- removeParticipant(struct VaultsStorageLibrary.VaultsList _self, uint256 _vaultId, address _participant) (external)
- createReward(struct VaultsStorageLibrary.RewardsList _self, uint256 _vaultId, address _winner, uint256 _yield) → uint256 (external)
- setVaultDuration(struct VaultsStorageLibrary.VaultsList _self, uint256 _vaultId, uint256 _duration) (external)
- isVaultPublic(struct VaultsStorageLibrary.VaultsList _self, uint256 _vaultId) → bool isPublic (external)
- isVaultRecurrent(struct VaultsStorageLibrary.VaultsList _self, uint256 _vaultId) → bool isRecurrent (external)
- get(struct VaultsStorageLibrary.VaultsList _self, uint256 _vaultId) → struct VaultsStorageLibrary.Vault (external)
- getParticipants(struct VaultsStorageLibrary.VaultsList _self, uint256 _vaultId) → address[] participants (external)
- getReward(struct VaultsStorageLibrary.RewardsList _self, uint256 _vaultId, uint256 _rewardId) → address winner, uint256 yield, uint256 blockNumberUpdated (external)
- getRewards(struct VaultsStorageLibrary.RewardsList _self, uint256 _vaultId) → struct VaultsStorageLibrary.Reward[] rewards (external)
- updateState(struct VaultsStorageLibrary.VaultsList _self, uint256 _vaultId, enum VaultsStorageLibrary.VaultState newState) (internal)
- updateCreatedBlock(struct VaultsStorageLibrary.VaultsList _self, uint256 _vaultId) (internal)
- updateLockDuration(struct VaultsStorageLibrary.VaultsList _self, uint256 _vaultId, uint256 _lockDuration) (internal)
- increaseBalance(struct VaultsStorageLibrary.VaultsList _self, uint256 _vaultId, uint256 _amount) (internal)
- decreaseBalance(struct VaultsStorageLibrary.VaultsList _self, uint256 _vaultId, uint256 _amount) (internal)
- increasePrizes(struct VaultsStorageLibrary.VaultsList _self, uint256 _vaultId, uint256 _amount) (internal)
- decreasePrizes(struct VaultsStorageLibrary.VaultsList _self, uint256 _vaultId, uint256 _amount) (internal)

Vault

☐ enum VaultsStorageLibrary.VaultState state
☐ address creator
☐ uint256 blockNumberCreated
☐ address lastUpdatedBy
☐ uint256 blockNumberUpdated
☐ uint256 minimumParticipants
☐ uint256 participantsNumber
☐ uint256 duration
☐ uint256 balance
☐ uint256 prizesAmount
☐ uint256 lockDuration
☐ bool isRecurrent
☐ bool isPublic

VaultsList

- □ mapping(uint256 => struct VaultsStorageLibrary.Vault) vaults
- □ mapping(uint256 => address[]) participants

Reward

- □ address winner
- □ uint256 yield
- □ uint256 blockNumberUpdated

RewardsList

- □ mapping(uint256 => struct VaultsStorageLibrary.Reward[]) rewards

VaultState

VaultReserve

Description

Contract in charge of holding usdc tokens and depositing them to Yield Aggregator

- `constructor()` (public)
- `initialize(address _usdcTokenAddress, address _aUsdcTokenAddress, address _backendAddress, address _conversionPoolAddress)` (external)

Initialize init the contract with the following parameters

This function is called only once during the contract initialization

- `deposit(uint256 _amount)` (public)

Transfers "_amount" of USDC tokens to Vault Smart Contract

- `moveFundsToAggregator(uint256 amount_usdc)` (external)

Transfers "amount_usdc" of tokens to Yield Aggregator contract

- `setBackendAddress(address _backendAddress)` (external)

Used to change the address of the Chainnlink Defender

- `setYieldSourceAddress(address _yieldSourceAddress)` (external)

Used to change the address of the contract responsible with yielding

- `depositUsdcThroughConversionPool(uint256 amount_usdc)` (external)
- `redeemAUsdcFromConversionPool(uint256 amount_ausdc)` (external)
- `transfer(address _account, uint256 _amount)` (external)

Used to deposit USDC to a destination address using funds from smart contract

- `pause()` (public)
- `unpause()` (public)
- `version() → string` (public)

FundsMovedToAggregator(uint256 amount_usdc)

☐ Emitted when USDC tokens are moved to Yield Aggregator contract

DepositToVault(address _depositant, uint256 _amount)

☐ Emitted when USDC tokens are moved to Yield Aggregator contract

YieldAggregator

Description

Interacts with Eth Anchor Smart Contract for token yielding

whiteListedOnly()

- `constructor()` (public)
- `initialize(address _ustTokenAddress, address _aUstToken, address _usdcTokenAddress, address _backendAddress, address _ethAnchorRouterAddress, address _swapperAddressContract)` (external)

Initialize init the contract with the following parameters this function is called only once during the contract initialization

- `deposit(uint256 _amount)` (external)

Deposits USDC tokens to this contract

Anyone can deposit funds to the Yield Source

- `startRedeemOfaUST(uint256 _amount)` (public)

Initiates withdraw of wUST (plus accrued interest) by claiming aUST tokens from EthAnchor

EthAnchor will send wUST at unspecified time to Yield Aggregator

- `swapToUsdcSendToVault()` (public)

All balance is wUST tokens is converted to USDC and sent to Vault Reserve

wUST is received an unspecified amount of time after redeeming aUST from EthAnchor

Once wUST arrives at Yield Aggregator, all of it is sent to Vault after swapping

- `setVaultReserveAddress(address _vaultReserveAddressContract)` (external)

Used to change the address of the Swapper Contract

- `setSwapperContract(address _swapperAddressContract)` (external)

Used to change the address of the Swapper Contract

Swapper Contracts: SwapCurve.sol (mainnet) or UniswapRouter.sol (testnet)

- `setBackendAddress(address _backendAddress)` (external)

Used to change the address of the Vault contract

- `setWhitelisting(bool _whitelisting)` (external)

Used to enable or disable the whitelisting

- `addToWhitelist(address _account)` (public)

Adds an account to whitelist

- `removeFromWhitelist(address _account)` (public)

Removes an account from whitelist

- `isWhitelisted(address _account) → bool` (public)

Checks whether an account is within whitelist or not

- `version() → string` (public)

Deposit(address account, uint256 amount_usdc, uint256 amount_ust)

□ Emitted when a user deposits USDC tokens

InitWithdraw(address account, uint256 amount_aust)

□ Emitted when a withdraw from Eth Anchor is initiated

FinishWithdraw(address _toAccount, uint256 amount_ust, uint256 amount_usdc)

- Emitted when the Yield Aggregator finishes the initiated withdraw

Findings

🟥🟥🟥🟥 Critical

The prize is not set.

The function `setPrize` actually updates the `_rewards` variable, as well as the `setReward` one. This leads us to the `_prizes` variable is never updated, so the `prizeOf` function will always return **0**.

Contracts: MultiToken.sol

Function: setPrize

Recommendation: set the `_prizes` variable in the `setPrize` function.

Status: Fixed

🟨🟨🟨 High

No high severity issues were found.

🟩🟩 Medium

Some tests failed

Going through testing is no additional description of running tests. Running by default returns 14 tests to be failed of 77.

Scope: testing

Recommendation: ensure all tests are running successfully, as well as there is at least 95% of coverage for statements and up to 100% for code branches.

Status: Not Fixed. **12** out of **65** are failing.

🟦 Low

1. **Unused function argument.**

   Arguments `_minAmountOut` and `_beneficiary` are never used.

   Contracts: SwapCurve.sol

   Function: swapToken

   Recommendation: remove unused arguments. Of course, the function is overridden, so it may want to remove only the names of arguments leaving the types in the declaration.

   Status: Fixed

2. **Not emitting events**

   Changing the crucial state of contracts requires emitting events. This allows the community to track such changes off-chain.

   Contract: YieldAggregator.sol

Functions: setVaultReserveAddress, setSwapperContract, setBackendAddress, setWhitelisting, addToWhitelist, removeFromWhitelist

Recommendation: emit events on changing important state variables.

Status: Fixed

3. **Not emitting events**

Changing the crucial state of contracts requires emitting events. This allows the community to track such changes off-chain.

Contract: VaultReserve.sol

Functions: setBackendAddress, setYieldSourceAddress

Recommendation: emit events on changing important state variables.

Status: Fixed

4. **Setting mapping value to false**

Instead of manually setting the boolean value in the mapping to **false**, it is better to delete that key from the mapping simply.

Contract: YieldAggregator.sol

Function: removeFromWhitelist

Recommendation: delete from mapping instead of setting to false.

Status: Fixed

5. **NetSpec displaced**

Function’s NetSpec block was placed in the block of the previous function.

Contract: VaultManager.sol

Function: distributeRewards

Recommendation: fix the NetSpec block.

Status: Fixed

Recommendations

1. Ensure all tests are successfully executed and cover at least 95% of statements and 100% of code branches.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.

Appendix A. Smart Contracts Security Issues

Category	Test Name	Result	Details
SWC-136	Unencrypted Private Data On-Chain	Passed	
SWC-135	Code With No Effects	Passed	
SWC-134	Message call with hardcoded gas amount	Passed	
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Passed	
SWC-132	Unexpected Ether balance	Passed	
SWC-131	Presence of unused variables	Passed	
SWC-130	Right-To-Left-Override control character (U+202E)	Passed	
SWC-129	Typographical Error	Passed	
SWC-128	DoS With Block Gas Limit		Not applicable in Solidity 0.8.x
SWC-127	Arbitrary Jump with Function Type Variable	Passed	
SWC-126	Insufficient Gas Griefing		Not applicable in Solidity 0.8.x
SWC-125	Incorrect Inheritance Order	Passed	
SWC-124	Write to Arbitrary Storage Location	Passed	
SWC-123	Requirement Violation	Passed	
SWC-122	Lack of Proper Signature Verification	Passed	
SWC-121	Missing Protection against Signature Replay Attacks	Passed	
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed	
SWC-119	Shadowing State Variables Function Default Visibility	Passed	
SWC-118	Incorrect Constructor Name	Passed	
SWC-117	Signature Malleability	Passed	
SWC-116	Block values as a proxy for time	Passed	
SWC-115	Authorization through tx.origin	Passed	
SWC-114	Transaction Order Dependence	Passed	
SWC-113	DoS with Failed Call	Passed	

SWC-112	Delegatecall to Untrusted Callee	Passed	
SWC-111	Use of Deprecated Solidity Functions	Passed	
SWC-110	Assert Violation	Passed	
SWC-109	Uninitialized Storage Pointer		Not applicable in Solidity 0.8.x
SWC-108	State Variable Default Visibility	Passed	
SWC-107	Reentrancy	Passed	
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed	
SWC-105	Unprotected Ether Withdrawal	Passed	
SWC-104	Unchecked Call Return Value	Passed	
SWC-103	Floating Pragma	Not passed	In Solidity 0.8.x floating pragma is not considered as a vulnerability
SWC-102	Outdated Compiler Version	Not passed	Version 0.8.3 has known bugs that do not affect the project but using version 0.8.9+ is recommended.
SWC-101	Integer Overflow and Underflow		Not applicable in Solidity 0.8.x
SWC-100	Function Default Visibility	Passed	