HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: RAMP

Date: April 9th, 2021

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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 fixed - upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for RAMP
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Complex
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/RAMP-DEFI/ramp-protocol
Commit	
Deployed contract	
Timeline	22 MAR 2021– 09 APR 2021
Changelog	09 APR 2021 – INITIAL AUDIT

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by RAMP (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer's smart contract and its code review conducted between March 22nd, 2021 – April 9th, 2021.

Scope

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

Repository: https://github.com/RAMP-DEFI/ramp-protocol
File:

- AppSettings.sol
- Controller.sol
- CakeLpStrategy.sol
- PancakePoolStrategy.sol
- StaticErcStrategy.sol
- SushiLpStrategy.sol
- BaseStrategy.sol
- RampStakingStrategy.sol
- ERC677.sol
- ERC677Receiver.sol
- ERC677Upgradeable.sol
- IERC677.sol
- IERC677Upgradeable.sol
- RToken.sol
- RUSD.sol
- Bank.sol
- BankV2.sol
- BonusPool.sol
- Vault.sol
- VaultV2.sol

We have scanned these smart contracts 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 ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

	▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are secure.

Insecure	Poor secured	Secured	Well-secured

You are here

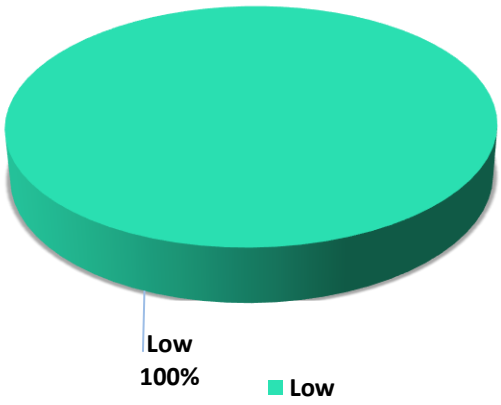
Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section, and all found issues can be found in the Audit overview section.

Security engineers found **2** low issue during the audit.

Notice: The source code of the contracts does not contain critical issues, well designed, and covered with tests. There are some minor issues about gas usage and logical optimisation, but they have no influence for the contracts' security.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Graph 1. The distribution of vulnerabilities after the first review.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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 can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

AS-IS overview

Controller.sol

Description

Controller is used by admins), for adding/removing tokens, strategy, for setting different states for strategy/vault

Imports

Controller has following imports:

- import "../dependencies/openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
- import "../dependencies/openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol";
- import "../dependencies/openzeppelin/contracts-upgradeable/token/ERC20/SafeERC20Upgradeable.sol";
- import "../dependencies/openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol";
- import "../dependencies/openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
- import "../strategies/BaseStrategy.sol";
- import "../libraries/RTokenAdapter.sol";
- import "../libraries/Helpers.sol";
- import "../token/RToken.sol";
- import "../interfaces/ramp/IPriceOracle.sol";
- import "../Bank.sol";
- import "../Vault.sol";

Inheritance

Controller inherit Helpers, AccessControlUpgradeable, ReentrancyGuardUpgradeable.

Usages

Controller contract has following usages:

- using SafeERC20Upgradeable for IERC20Upgradeable;
- using RTokenAdapter for IERC20Upgradeable;

Structs

Controller contract has no custom structures.

Enums

Controller contract has no enums.

Events

Controller contract has following events:

- event TokenAdded(address token, uint16 collateralRatio, uint256 mintCapacity, address oracle, address strategy);
- event CollateralRatioUpdated(address token, uint16 collateralRatio);
- event LiquidationRatioUpdated(address token, uint16 _liquidationRatio);
- event MintCapacityUpdated(address token, uint256 _mintCapacity);
- event TreasuryUpdated(address _treasury);
- event StrategyInstalled(address token, address strategy);
- event StrategyUnInstalled(address token, address strategy);
- event StrategyWithdrawal(address token, address strategy);
- event EmergencyStrategyWithdrawal(address token, address strategy, bool abandonRewards);
- event StrategyPaused(address token, address strategy);
- event OracleUpdated(address token, address oracle);

Modifiers

Controller has following modifiers:

- onlyOperator ()

Fields
Controller contract has following fields and constants:

- Bank public bank;
- Vault public vault;
- mapping(address => address) public rTokensToAssets;

Functions
Controller has following public functions:

- *addToken*
- *setOracle*
- *updateCollateralRatio*
- *updateLiquidationRatio*
- *updateMintCapacity*
- *updateTreasury*
- *activateStrategy*
- *uninstallStrategy*
- *withdrawStrategy*
- *replaceStrategy*
- *emergencywithdrawStrategy*
- *pauseStrategy*
- *updateStrategyStatus*

AppSettings.sol

Description

Simple contract to keep system-wide settings.

Imports

StakeManager has following imports:

- /openzeppelin/contracts/access/AccessControl.sol

Inheritance
AppSettings is AccessControl.

Usages

AppSettings contract has no usages.

Structs

AppSettings contract has no data structures

Enums

AppSettings contract has no enums.

Events

AppSettings contract has no events.

Modifiers

AppSettings has following modifiers:

- onlyAdmin ()

Fields
AppSettings contract has following fields and constants:

- mapping(bytes32 => uint256) public uintStorage;
- mapping(bytes32 => string) public stringStorage;
- mapping(bytes32 => address) public addressStorage;
- mapping(bytes32 => bool) public boolStorage;

Functions
AppSettings has following public functions:

- *constructor*
- *setUint*
- *setString*
- *setAddress*
- *setBool*

CakeLpStrategy.sol

Description

Pluggable contracts that allow for (re)investing of tokens from the Vaults

Imports

CakeLpStrategy has following imports:

- import "../../dependencies/openzeppelin/contracts-upgradeable/math/SafeMathUpgradeable.sol";
- import "../../dependencies/openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol";
- import "../../dependencies/openzeppelin/contracts-upgradeable/token/ERC20/SafeERC20Upgradeable.sol";
- import "../../dependencies/openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
- import "../../strategies/BaseStrategy.sol";
- import "../../interfaces/pancake/IMasterChef.sol";
- import "../../interfaces/pancake/IPancakeswapRouter.sol";

Inheritance

CakeLpStrategy is OwnableUpgradeable, BaseStrategy.

Usages

CakeLpStrategy contract has following usages:

- using SafeMathUpgradeable for uint256;
- using SafeERC20Upgradeable for IERC20Upgradeable;

Structs

CakeLpStrategy contract has following data structures:

- PoolInfo

Enums

CakeLpStrategy contract has no enums.

Events

CakeLpStrategy contract has following events:

- SetPoolInfo
- ChangedRampPerBlock

- EmptyRewardPool

Modifiers

CakeLpStrategy has no modifiers

Fields

CakeLpStrategy contract has following fields and constants:

- mapping(address => bool) private recoverableTokensBlacklist;

Functions
CakeLpStrategy has following public functions:

- *initializer*
- *getPoolAmount*
- *setPoolInfo*
- *getBalance*
- *getStrategyType*
- *update*
- *onDeposit*
- *onWithdraw*
- *work*
- *emergencyWithdraw*
- *sweep*

PancakePoolStrategy.sol

Description

Pluggable contracts that allow for (re)investing of tokens from the Vaults

Imports

PancakePoolStrategy has following imports:

- import "../../dependencies/openzeppelin/contracts-upgradeable/math/SafeMathUpgradeable.sol";
- import "../../dependencies/openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol";

- import "../../dependencies/openzeppelin/contracts-upgradeable/token/ERC20/SafeERC20Upgradeable.sol";
- import "../../dependencies/openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
- import "../../strategies/BaseStrategy.sol";
- import "../../interfaces/pancake/IMasterChef.sol";
- import "../../interfaces/pancake/ISmartChef.sol";
- import "../../interfaces/pancake/IPancakeswapRouter.sol";
- import "../../strategies/BaseStrategy.sol";

Inheritance
PancakePoolStrategy is OwnableUpgradeable, BaseStrategy.

Usages

PancakePoolStrategy contract has following usages:

- using SafeMathUpgradeable for uint256;
- using SafeERC20Upgradeable for IERC20Upgradeable;

Structs

PancakePoolStrategy contract has following data structures:

- PoolInfo
- HarvestSignature

Enums

PancakePoolStrategy contract has no enums.

Events

PancakePoolStrategy contract has no events:

Modifiers

PancakePoolStrategy has no modifier

Fields

PancakePoolStrategy contract has following fields and constants:

- address constant public wbnb = address(0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c);
- address constant public cake = address(0x0E09FaBB73Bd3Ade0a17ECC321fD13a19e81cE82);
- uint256 public currentPool;
- PoolInfo[] public poolInfo;
- uint256[] public yields;
- address constant public pancakeRouter = address(0x05fF2B0DB69458A0750badebc4f9e13aDd608C7F);
- address constant public masterchef = address(0x73feaa1eE314F8c655E354234017bE2193C9E24E);
- address public yieldCalculator;
- uint constant public MAX_FEE = 100;
- uint constant public WITHDRAWAL_FEE = 10;
- uint constant public WITHDRAWAL_MAX = 10000;
- address[] public wbnbToCakeRoute;
- string public constant name = "RampPancakePoolStrategy";
- uint256 public nonce;

Functions

PancakePoolStrategy has following public functions:

- *initializer*
- *deposit*
- *onDeposit*
- *work*
- *getBalance*
- *getBalance*
- *update*
- *onWithdraw*
- *onLiquidate*
- *harvest*
- *getPoolAmount*

- *balanceOfCake*
- *poolBalance*
- *balanceOf*
- *emergencyWithdraw*
- *getStrategyType*
- *sweep*

StaticErcStrategy.sol

Description

Pluggable contracts that allow for (re)investing of tokens from the Vaults

Imports

StaticErcStrategy has following imports:

- import "../../dependencies/openzeppelin/contracts-upgradeable/math/SafeMathUpgradeable.sol";
- import "../../dependencies/openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol";
- import "../../dependencies/openzeppelin/contracts-upgradeable/token/ERC20/SafeERC20Upgradeable.sol";
- import "../../dependencies/openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
- import "../../strategies/BaseStrategy.sol";
- import "hardhat/console.sol";

Inheritance

StaticErcStrategy is OwnableUpgradeable, BaseStrategy.

Usages

StaticErcStrategy contract has following usages:

- using SafeMathUpgradeable for uint256;
- using SafeERC20Upgradeable for IERC20Upgradeable;

Structs

StaticErcStrategy contract has following data structures:

- PoolInfo
- UserInfo

Enums

StaticErcStrategy contract has no enums.

Events

StaticErcStrategy contract has no events:

Modifiers

StaticErcStrategy has no modifier

Fields

StaticErcStrategy contract has following fields and constants:

- PoolInfo public poolInfo;
- uint256 private poolAmount;
- mapping(address => UserInfo) public userInfo;
- IERC20Upgradeable rewardToken;
- IERC20Upgradeable stakedToken;
- address public farmingWallet;

Functions

StaticErcStrategy has following public functions:

- *initializer*
- *deposit*

- *onDeposit*
- *work*
- *getBalance*
- *getBalance*
- *update*
- *onWithdraw*
- *onLiquidate*
- *harvest*
- *getPoolAmount*
- *balanceOfCake*
- *poolBalance*
- *balanceOf*
- *emergencyWithdraw*
- *getStrategyType*
- *sweep*

SushiLpStrategy.sol

Description

Pluggable contracts that allow for (re)investing of tokens from the Vaults

Imports

SushiLpStrategy has following imports:

- import "../../dependencies/openzeppelin/contracts-upgradeable/math/SafeMathUpgradeable.sol";
- import "../../dependencies/openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol";
- import "../../dependencies/openzeppelin/contracts-upgradeable/token/ERC20/SafeERC20Upgradeable.sol";
- import "../../dependencies/openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";

- import "../../strategies/BaseStrategy.sol";
- import "hardhat/console.sol";

Inheritance
SushiLpStrategy is OwnableUpgradeable, BaseStrategy.

Usages

SushiLpStrategy contract has following usages:

- using SafeMathUpgradeable for uint256;
- using SafeERC20Upgradeable for IERC20Upgradeable;

Structs

SushiLpStrategy contract has following data structures:

- UserInfo

Enums

SushiLpStrategy contract has no enums.

Events

SushiLpStrategy contract has no events:

- event StrategyDeposit(address indexed user, uint256 indexed pid, uint256 amount);
- event StrategyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);
- event SetPoolInfo(address token, uint256 poolId);
- event Liquidated(address indexed user, uint256 pid, uint256 _amount);

Modifiers

SushiLpStrategy has no modifier

Fields

SushiLpStrategy contract has following fields and constants:

- PoolInfo public poolInfo;

- uint256 private poolAmount;
- mapping(address => UserInfo) public userInfo;
- IERC20Upgradeable rewardToken;
- IERC20Upgradeable stakedToken;
- address public farmingWallet;

Functions

SushiLpStrategy has following public functions:

- *initializer*
- *deposit*
- *onDeposit*
- *work*
- *getBalance*
- *getBalance*
- *update*
- *onWithdraw*
- *onLiquidate*
- *harvest*
- *getPoolAmount*
- *balanceOfCake*
- *poolBalance*
- *balanceOf*
- *emergencyWithdraw*
- *getStrategyType*
- *sweep*

RampStakingStrategy.sol

Description

Pluggable contracts that allow for (re)investing of tokens from the Vaults

Imports

RampStakingStrategy has following imports:

- import "../dependencies/openzeppelin/contracts-upgradeable/math/SafeMathUpgradeable.sol";
- import "../dependencies/openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol";
- import "../dependencies/openzeppelin/contracts-upgradeable/token/ERC20/SafeERC20Upgradeable.sol";
- import "../dependencies/openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
- import "../strategies/BaseStrategy.sol";

Inheritance
RampStakingStrategy is OwnableUpgradeable, BaseStrategy.

Usages

RampStakingStrategy contract has following usages:

- using SafeMathUpgradeable for uint256;
- using SafeERC20Upgradeable for IERC20Upgradeable;

Structs

RampStakingStrategy contract has following data structures:

- PoolInfo

Enums

RampStakingStrategy contract has no enums.

Events

RampStakingStrategy contract has no events:

- event ChangedRampPerBlock(uint256 oldRampPerBlock, uint256 newRampPerBlock)
- event EmptyRewardPool()

Modifiers

RampStakingStrategy has no modifier

Fields

RampStakingStrategy contract has following fields and constants:

- uint256 constant DECIMALS = 18;
- uint256 constant UNITS = 10 ** DECIMALS;
- uint256 public constant BLOCK_ESTIMATE = 2425847;
- IERC20Upgradeable public rampToken;
- address public rampTokenFarmingWallet;
- PoolInfo public poolInfo;
- uint256 private poolAmount;

Functions

SushiLpStrategy has following public functions:

- *initializer*
- *deposit*
- *onDeposit*
- *work*
- *getBalance*
- *getBalance*
- *update*
- *onWithdraw*
- *onLiquidate*
- *harvest*
- *getPoolAmount*
- *balanceOfCake*
- *poolBalance*
- *balanceOf*
- *emergencyWithdraw*
- *getStrategyType*
- *sweep*

Bank.sol

Description

Allows user to borrow rUSD, to repay the borrowed rUSD, prices can come from Oracles or from our own offchain oracle that provides signed data, allows to fetch the getInterestDue for every user, liquidate funds

Imports

Bank has following imports:

- import "./token/RUSD.sol";
- import "./interfaces/ramp/IPriceOracle.sol";
- import "./strategies/BaseStrategy.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/math/SafeMathUpgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/token/ERC20/SafeERC20Upgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/cryptography/ECDSAUpgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
- import "./libraries/PriceReceiver.sol";
- import "./libraries/RayMath.sol";
- import "./token/RToken.sol";
- import "./Vault.sol";
- import "./libraries/Helpers.sol";

Inheritance

Bank is Initializable, AccessControlUpgradeable, Helpers, ReentrancyGuardUpgradeable, PriceReceiver

Usages

Bank contract has following usages:

- using SafeMathUpgradeable for uint256;
- using SafeERC20Upgradeable for IERC20Upgradeable;
- using ECDSAUpgradeable for bytes32;
- using RayMath for uint256;

Structs

Bank contract has no data structures.

Enums

Bank contract has no enums.

Events

Bank contract has no events:

- TokenUpdated
- Borrow
- InterestChanged
- Repay
- Liquidated
- TokenAdded

Modifiers

Bank has following modifiers:

- onlyEOA
- onlyController
- onlyOperator

Fields

Bank contract has following fields and constants:

- uint256 internal constant SECONDS_PER_YEAR = 365 days;
- mapping(address => BankTokenInfo) public tokens;

- Vault vault;
- RUSD rUSD;
- address treasury;

Functions

Bank has following public functions:

- *borrow*
- *getInterestRate*
- *getRepayQuote*
- *getInterestDue*
- *getMinCollateralRatio*
- *getPriceFromOracle*
- *getInterestFromOracle*
- *repay*
- *liquidate*
- *getBorrowed*
- *getMaxBorrowable*
- *isLiquidatable*
- *setMinCollateralRatio*
- *setMintCapacity*
- *setLiquidationRatio*
- *setOracle*
- *setTreasury*
- *addToken*
- *updateToken*
- *setPriceSigner*
- *initializeController*

BankV2.sol

Description

Allows user to borrow rUSD, to repay the borrowed rUSD, prices can come from Oracles or from our own offchain oracle that provides signed data, allows to fetch the getInterestDue for every user, liquidate funds

Imports

BankV2 has following imports:

- import "./token/RUSD.sol";
- import "./interfaces/ramp/IPriceOracle.sol";
- import "./strategies/BaseStrategy.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/math/SafeMathUpgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/token/ERC20/SafeERC20Upgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/cryptography/ECDSAUpgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
- import "./libraries/PriceReceiver.sol";
- import "./libraries/RayMath.sol";
- import "./token/RToken.sol";
- import "./Vault.sol";
- import "./libraries/Helpers.sol";

Inheritance

BankV2 is Initializable, AccessControlUpgradeable, Helpers, ReentrancyGuardUpgradeable, PriceReceiver

Usages

BankV2 contract has following usages:

- using SafeMathUpgradeable for uint256;
- using SafeERC20Upgradeable for IERC20Upgradeable;
- using ECDSAUpgradeable for bytes32;

- using RayMath for uint256;

Structs

BankV2 contract has no data structures.

Enums

BankV2 contract has no enums.

Events

BankV2 contract has no events:

- TokenStatusUpdated
- Borrow
- InterestChanged
- Repay
- Liquidated
- TokenAdded

Modifiers

BankV2 has following modifiers:

- onlyEOA
- onlyController
- onlyOperator

Fields

BankV2 contract has following fields and constants:

- uint256 internal constant SECONDS_PER_YEAR = 365 days;
- mapping(address => BankTokenInfo) public tokens;
- Vault vault;
- RUSD rUSD;
- address treasury;

Functions

BankV2 has following public functions:

- *borrow*
- *getInterestRate*
- *getRepayQuote*
- *getInterestDue*
- *getMinCollateralRatio*
- *getPriceFromOracle*
- *getInterestFromOracle*
- *repay*
- *liquidate*
- *getBorrowed*
- *getMaxBorrowable*
- *isLiquidatable*
- *setMinCollateralRatio*
- *setMintCapacity*
- *setLiquidationRatio*
- *setOracle*
- *setTreasury*
- *addToken*
- *updateTokenStatus*
- *setPriceSigner*

BonusPool.sol

Description

Allow the distribution of additional RAMP rewards to users who staked rTOKENs into rMINT

Imports

BonusPool has following imports:

- "./dependencies/openzeppelin/contracts/math/SafeMath.sol";
- "./dependencies/openzeppelin/contracts/token/ERC20/SafeERC20.sol";
- "hardhat/console.sol";

- "./dependencies/openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";

Inheritance
BonusPool is AccessControlUpgradeable

Usages

BonusPool contract has following usages:

- using SafeMath for uint256;
- using SafeERC20 for IERC20;

Structs

BonusPool contract has following data structures:

- PoolInfo
- UserInfo

Enums

BonusPool contract has no enums.

Events

BonusPool contract has no events:

- Claimed
- EmergencyWithdraw

Modifiers

BonusPool has following modifiers:

- onlyOperator
- onlyVault

Fields

BonusPool contract has following fields and constants:

- bytes32 public constant VAULT_ROLE = keccak256("VAULT_ROLE");
- bytes32 public constant OPERATOR_ROLE = keccak256("OPERATOR_ROLE");
- uint256 constant UNITS = 10e18;
- mapping(address => mapping(address => UserInfo)) public userInfo;
- IERC20 public rewardToken;
- address public farmingWallet;
- address public devAddress;
- mapping(address => PoolInfo) public poolInfo;
- uint256 public totalRewardsPerBlock;
- uint256 public startBlock;

Functions

BonusPool has following public functions:

- *initialize*
- *add*
- *setRewardsPerBlock*
- *updateRewards*
- *updatePoolUser*
- *getReward*
- *claimReward*
- *getPoolReward*
- *setDevAddress*

Vault.sol

Description

Provide ability to deposit and withdraw funds to strategies and basic funds management functionality.

Imports

Vault has following imports:

- import "./token/RUSD.sol";

- import "./strategies/BaseStrategy.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/math/SafeMathUpgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/token/ERC20/SafeERC20Upgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/cryptography/ECDSAUpgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
- import "./libraries/RayMath.sol";
- import "./token/RToken.sol";
- import "./libraries/PriceReceiver.sol";
- import "./libraries/Helpers.sol";
- import "./Bank.sol";
- import "./interfaces/ramp/IPriceInfo.sol";
- import "./BonusPool.sol";
- import "./controllers/Controller.sol";
- import "hardhat/console.sol";

Inheritance

Vault is ERC677Receiver, AccessControlUpgradeable, ReentrancyGuardUpgradeable, Helpers, PriceReceiver

Usages

Vault contract has following usages:

- using SafeMathUpgradeable for uint256;
- using SafeERC20Upgradeable for IERC20Upgradeable;
- using ECDSAUpgradeable for bytes32;
- using RayMath for uint256;

Structs

Vault contract has following data structures:

- WithdrawVariables

Enums

Vault contract has no enums.

Events

Vault contract has no events:

- TokenUpdated
- Withdraw
- TokenAdded
- WithdrawLiquidated
- Staked
- Unstaked
- Deposit

Modifiers

Vault has following modifiers:

- onlyBank
- onlyController
- onlyOperator
- onlyEOA

Fields

Vault contract has following fields and constants:

- uint256 internal constant SECONDS_PER_YEAR = 365 days;
- uint256 internal constant BONUSPOOL_DEFAULT_REWARDPERBLOCK = 0;
- RUSD rUSD;
- Bank bank;

- BonusPool bonusPool;
- address public controller;
- address public treasury;
- mapping(address => VaultTokenInfo) public tokens;

Functions

Vault has following public functions:

- *initialize*
- *initializeBank*
- *initializeController*
- *onTokenTransfer*
- *deposit*
- *getCollateralizableRToken*
- *withdraw*
- *onRepay*
- *onLiquidate*
- *getMaxWithdrawable*
- *recallFromStrategy*
- *reinvestStrategy*
- *getAssetForRToken*
- *getAssetBalance*
- *getRTokenBalance*
- *getPoolBalance*
- *withdrawLiquidated*
- *getStrategyPoolSize*
- *stake*
- *unstake*
- *getMaxUnstakeable*
- *_unstake*
- *addToken*
- *updateToken*
- *tokenExists*
- *getToken*
- *setPriceSigner*
- *setBonusPool*
- *setTreasury*

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

VaultV2.sol

Description

Provide ability to deposit and withdraw funds to strategies and basic funds management functionality.

Imports

VaultV2 has following imports:

- import "./token/RUSD.sol";
- import "./strategies/BaseStrategy.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/utils/ReentrancyGuardUpgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/math/SafeMathUpgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/token/ERC20/SafeERC20Upgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/cryptography/ECDSAUpgradeable.sol";
- import "./dependencies/openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
- import "./libraries/RayMath.sol";
- import "./token/RToken.sol";
- import "./libraries/PriceReceiver.sol";
- import "./libraries/Helpers.sol";
- import "./Bank.sol";
- import "./interfaces/ramp/IPriceInfo.sol";
- import "./BonusPool.sol";
- import "./controllers/Controller.sol";
- import "hardhat/console.sol";

Inheritance

VaultV2 is ERC677Receiver, AccessControlUpgradeable, ReentrancyGuardUpgradeable, Helpers, PriceReceiver

Usages

VaultV2 contract has following usages:

- using SafeMathUpgradeable for uint256;
- using SafeERC20Upgradeable for IERC20Upgradeable;
- using ECDSAUpgradeable for bytes32;
- using RayMath for uint256;

Structs

VaultV2 contract has following data structures:

- PoolInfo
- UserInfo

Enums

VaultV2 contract has no enums.

Events

VaultV2 contract has no events:

- TokenStatusUpdated
- Withdraw
- TokenAdded
- WithdrawLiquidated
- Staked
- Unstaked
- Deposit

Modifiers

VaultV2 has following modifiers:

- onlyBank
- onlyController

- onlyOperator
- onlyEOA

Fields

VaultV2 contract has following fields and constants:

- uint256 internal constant SECONDS_PER_YEAR = 365 days;
- uint256 internal constant BONUSPOOL_DEFAULT_REWARDPERBLOCK = 0;
- RUSD rUSD;
- Bank bank;
- BonusPool bonusPool;
- address public controller;
- address public treasury;
- mapping(address => VaultTokenInfo) public tokens;

Functions

VaultV2 has following public functions:

- *initialize*
- *initializeBank*
- *initializeController*
- *onTokenTransfer*
- *deposit*
- *getCollateralizableRToken*
- *withdraw*
- *onRepay*
- *onLiquidate*
- *getMaxWithdrawable*
- *recallFromStrategy*
- *reinvestStrategy*
- *getAssetForRToken*
- *getAssetBalance*
- *getRTokenBalance*
- *getPoolBalance*
- *withdrawLiquidated*
- *getStrategyPoolSize*

- *stake*
- *unstake*
- *getMaxUnstakeable*
- *_unstake*
- *addToken*
- *updateToken*
- *tokenExists*
- *getToken*
- *setPriceSigner*
- *setBonusPool*
- *setTreasury*

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Audit overview

Low

1. Bank.borrow and BankV2.borrow functions execute mint operations before changing the total count of the token supply and before the changing user account state in the contract. We strongly recommend you execute mint operation after the accounting.

2. Vault.withdraw and VaultV2.withdraw functions execute funds transfer operations before changing the funds accounting. We strongly recommend you execute transfer operation after changing the funds sum on the account.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract, high-level description of functionality was presented in As-Is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found **2** low issues during the audit.

Notice: The source code of the contracts does not contain critical issues, well designed, and covered with tests. There are some minor issues about gas usage and logical optimisation, but they have no influence for the contracts' security.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to 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 security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. 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 security of smart contracts.

Technical Disclaimer

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