SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Woonkly

Date: April 2nd, 2021

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

Document

Name	Smart Contract Code Review and Security Analysis Report for Woonkly
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Exchange
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/Woonkly/DEXsmartcontractsPreRelease
Commit	
Deployed contract	
Timeline	23 MAR 2021 – 1 APR 2021
Changelog	1 APR 2021 – INITIAL AUDIT 2 APR 2021 – ADDING CUSTOMER NOTICE

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Woonkly (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 23rd, 2021 – April 1st, 2021. Customer notices have been added on April 2nd.

Scope

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

Repository: https://github.com/Woonkly/DEXsmartcontractsPreRelease
File:

InvestorManager.sol
IwoonklyPOS.sol
IWStaked.sol
Pausabled.sol
StakeManager.sol
WonklyDEX.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 ▪ 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 secured.

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 **7** high, **7** low and **3** informational issue during the audit. All issues have been acknowledged/accepted by the Customer.

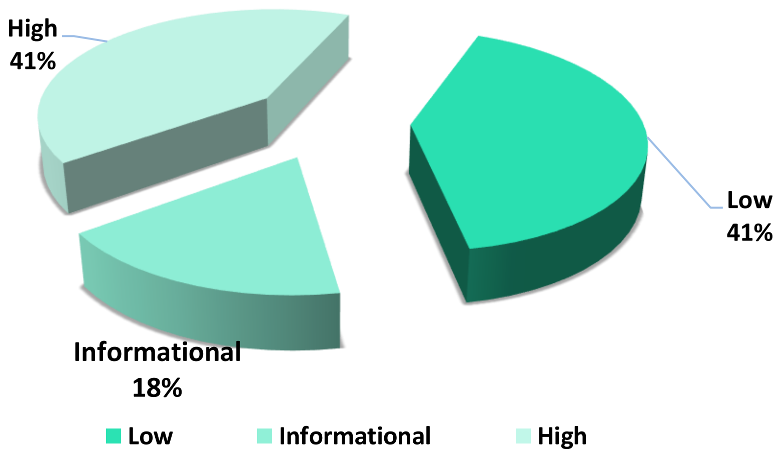
Notice:

1. It is much better to move reward and payment functionality to StakeManager From DEX. To get accounting and funds transferring operation in one contract and can incapsulate funds operations into functions like withdraw() and add stake(). It will give you the ability to minimize risks and follow the OOP development style.
2. WonklyDEX.coinToToken() & WonklyDEX.tokenToCoin() functions can be unsafe. Fees and WonklyDEX.op_eward calculation is not the part of user price calculation.

Customer notice: WonklyDEX.coinToToken() and WonklyDEX.tokenToCoin() always have the value with fee, inside the user value is already incorporated the cash fee.

3. The source code of the contract contains problems with architecture and code style. Gas consumption is not optimal.

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



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.

AS-IS overview

WonklyDEX.sol

Description

WonklyDEX provides basic dex operations.

Imports

WonklyDEX has following imports:

- https://github.com/Woonkly/OpenZeppelinBaseContracts/contracts/math/SafeMath.sol
- https://github.com/Woonkly/OpenZeppelinBaseContracts/contracts/token/ERC20/ERC20.sol
- https://github.com/Woonkly/OpenZeppelinBaseContracts/contracts/utils/ReentrancyGuard.sol
- https://github.com/Woonkly/DEXsmartcontractsPreRelease/StakeManager.sol
- https://github.com/Woonkly/DEXsmartcontractsPreRelease/IwoonklyPOS.sol
- https://github.com/Woonkly/MartinHSolUtils/PausabledLMH.sol
- https://github.com/Woonkly/DEXsmartcontractsPreRelease/IWStaked.sol

Inheritance

WonklyDEX inherit Owners, PausabledLMH, ReentrancyGuard.

Usages

WonklyDEX contract has following usages:

- SafeMath for uint.

Structs

WonklyDEX contract has following data structures:

- Stake – structure to store stakes.
- processRewardInfo — structure to store information about rewards

Enums

WonklyDEX contract has no enums.

Events

WonklyDEX contract has following events:

- event FeeChanged(uint32 oldFee, uint32 newFee);
- event BaseFeeChanged(uint32 oldbFee, uint32 newbFee);
- event GasRequiredChanged(uint256 oldg, uint256 newg);
- event OperationsChanged(address oldOp, address newOp);
- event WoonklyPOSChanged(address oldaddr, address newaddr);
- event BeneficiaryChanged(address oldBn, address newBn);
- event StakeAddrChanged(address old, address news);
- event StakeSHAddrChanged(address old, address news);
- event WoopSHFundsChanged(address old, address news);
- event CoinReceived(uint256 coins);
- event PoolCreated(uint256 totalLiquidity, address investor, uint256 token_amount);
- event PoolClosed(uint256 eth_reserve, uint256 token_reserve, uint256 liquidity, address destination);
- event InsuficientRewardFund(address account, bool isCoin, bool isSH);
- event NewLeftover(address account, uint256 leftover, bool isCoin);
- event PurchasedTokens(address purchaser, uint256 coins, uint256 tokens_bought);
- event FeeTokens(uint256 bnPart, uint256 liqPart, uint256 opPart, address beneficiary, address operations);
- event TokensSold(address vendor, uint256 eth_bought, uint256 token_amount);
- event FeeCoins(uint256 bnPart,uint256 liqPart,uint256 opPart,address beneficiary,address operations);
- event LiquidityChanged(uint256 oldLiq, uint256 newLiq, bool isSH);
- event LiquidityWithdraw(address investor, uint256 coins, uint256 token_amount, uint256 newliquidity, bool isSH);

Modifiers

WonklyDEX has no custom modifiers.

Fields

WonklyDEX contract has following fields and constants:

- IERC20 token;
- uint256 public totalLiquidity;
- uint32 internal _fee;
- uint32 internal _baseFee;
- address internal _operations;
- address internal _beneficiary;
- uint256 internal _coin_reserve;
- address internal _wooncky POS;
- address internal _wooncly BEP20;
- IWStaked internal _stakes;
- address internal _stakeable;
- IWStaked internal _stakesSH;
- address internal _stakeableSH;
- address internal _woopSharedFunds;
- uint256 internal _gasRequired;

Functions

WonklyDEX has following public view and pure functions:

- getCoinReserve
- getFee
- getBaseFee
- getGasRequired
- getOperations

- getWoonklyPOS
- getWoopSHFunds
- getBeneficiary
- getStakeAddr
- getStakeAddrSH
- getMyCoinBalance
- getMyTokensBalance
- getSCtokenAddress
- getCalcRewardAmount
- price
- planePrice
- calcDeal
- isOverLimit
- getPercImpact
- currentTokensToCoin
- getMaxAmountSwap
- currentCoinToTokens
- getSTK
- calcTokenToAddLiq
- getValuesLiqWithdraw
- getMaxValuesLiqWithdraw

WonklyDEX has following owner functions:

- setWoopSHFunds
- setStakeSHAddr
- setStakeAddr
- setBeneficiary
- setWoonklyPOS
- setOperations

- setGasRequired
- setBaseFee
- setFee
- closePool
- migratePool

WonklyDEX has following user functions:

- WithdrawLiquidity
- AddLiquidity
- tokenToCoin
- coinToToken
- WithdrawReward
- createPool
- fallback
- addCoin
- receive

StakeManager.sol

Description

StakeManager provides basic operations with stake.

Imports

StakeManager has following imports:

- https://github.com/Woonkly/OpenZeppelinBaseContracts/contracts/math/SafeMath.sol
- https://github.com/Woonkly/OpenZeppelinBaseContracts/contracts/token/ERC20/ERC20.sol
- https://github.com/Woonkly/MartinHSolUtils/Utils.sol
- https://github.com/Woonkly/MartinHSolUtils/Owners.sol

Inheritance

StakeManager inherit

- Owners
- ERC20.

Usages

StakeManager contract has following usages:

- SafeMath for uint.

Structs

StakeManager contract has following data structures:

- Stake – structure to store stakes.

Enums

StakeManager contract has no enums.

Events

StakeManager contract has following events:

- event addNewStake(address account, uint256 amount);
- event StakeAdded(address account, uint256 oldAmount, uint256 newAmount);
- event StakeReNewed(address account, uint256 oldAmount, uint256 newAmount);
- event RewaredChanged(address account, uint256 amount, bool isCoin, uint8 set);
- event StakeRemoved(address account);
- event StakeSubstracted(address account, uint256 oldAmount, uint256 subAmount, uint256 newAmount);
- event AllStakeRemoved();

Modifiers

StakeManager has following modifiers:

- onlyStakeExist
- onlyNewStake

- onlyStakeIndexExist

Fields

StakeManager contract has following fields and constants:

- uint256 internal _lastIndexStakes;
- mapping(uint256 => Stake) internal _Stakes;
- mapping(address => uint256) internal _IDStakesIndex;
- uint256 internal _StakeCount;

Functions

StakeManager has following public view and pure functions:

- getStakeCount
- getLastIndexStakes
- StakeExist
- StakeIndexExist
- getReward
- getStake
- getStakeByIndex
- getAllStake

StakeManager has following owner functions:

- newStake
- addToStake
- renewStake
- changeReward
- removeStake
- substractFromStake
- removeAllStake

InvestorManager.sol

Description

InvestorManager provides basic operations with investor.

Imports

StakeManager has following imports:

- https://github.com/Woonkly/OpenZeppelinBaseContracts/contracts/math/SafeMath.sol

Inheritance

InvestorManager no inheritance.

Usages

InvestorManager contract has following usages:

- SafeMath for uint.

Structs

InvestorManager contract has following data structures:

- Investor – structure to store stakes.

Enums

InvestorManager contract has no enums.

Events

InvestorManager contract has following events:

- event addNewInvestor(address account, uint256 liquidity);
- event removeInvestor(address account);

Modifiers

InvestorManager has following modifiers:

- onlyNewInvestor
- onlyInvestorExist
- onlyInvestorIndexExist

Fields

InvestorManager contract has following fields and constants:

- uint256 internal _lastIndexInvestors;
- mapping(uint256 => Investor) internal _Investors;
- mapping(address => uint256) internal _IDInvestorsIndex;
- uint256 internal _InvestorCount;

Functions

InvestorManager has following public view and pure functions:

- getInvestorCount
- InvestorExist
- InvestorIndexExist
- getInvestor
- getInvestorLiquidity
- getInvestorByIndex
- getAllInvestor

Audit overview

■■■ High

1. The owner can mint coins without any control to any address and accounting in InvestorManager.newStake(), InvestorManager.addToStake() functions.

Customer notice: InvestorManager is not used.

2. There are no contract tests. Contract functionality should be covered by tests.

3. The owner can manually change the rewards to any account in the pool in the StakeManager.changeReward() function.

4. The owner can remove each user stake without any funds withdrawal in the StakeManager.removeStake() function.

5. The safeERC20 library should be used to execute the transfer and other operations on the token.

Customer accepts this risk.

6. WonklyDEX.createPool() adds to the stake total amount of liquidity.

Customer notice: The coin part is added in this case BNB because it is liquidity concept. It is a relation between the amount of BNB and the amount of the woop token, but the liquidity is always in BNB.
Hacken notice: when you create the new pool for some address (create function is in public access) you use function *_addStake(_msgSender(), totalLiquidity, false);* where *totalLiquidity* is equal to *address(this).balance* not to *token_amount* value**,** which was received as function parameter.

7. WonklyDEX.closePool() function can be called by owners and move all funds to op address without any funds returning to users.

■ Low

1. InvestorManager.InvestorIndexExist(). Simplify code to reduce gas usage.

Customer notice: InvestorManager is not used.

2. InvestorManager._InvestorExist(). Simplify code to reduce gas usage.

Customer notice: InvestorManager is not used.

3. WonklyDEX.Price() can be changed to pure
4. WonklyDEX.planePrice() can be changed to pure
5. WonklyDEX.isOverLimit(). Should be simplified to gas usage reduce
6. There are no requirements for WonklyDEX.createPool() msg.value amount and there is no this value usage. Why does the function should be payable?
7. The allowance validation in functions WonklyDEX.createPool(), WonklyDEX.migratePool(), WonklyDEX.tokenToCoin(), WonklyDEX.AddLiquidity() are redundant. Because of using 'require' to transferFrom operations.

■ Lowest / Code style / Best Practice / Informational

1. Hardcoded values should be moved to constants.
2. Require error messages should be set and should describe the error reason.
3. Fee and BaseFee are different entities. If you want to emit events and build the system base on them, you should emit different events for changing these entities.

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 **7** high, **7** low and **3** informational issue during the audit. All issues have been acknowledged/accepted by the Customer.

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.