HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: TacoSwap

Date: March 3rd, 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 TacoSwap.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Token, Governance, TimeLock, Defi
Platform	Binance Smart Chain / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	
Commit	
Deployed contract	Masterchef: 0x36f44a1C8e973739D0034FF1B9B9f6c4c7085625 Timelock: 0xbF4e3A945de1BD5869A6EfdC9230F50c75757BC6 Token: 0x9066e87Bac891409D690cfEfA41379b34af06391
Timeline	1 MAR 2021 – 3 MAR 2021
Changelog	3 MAR 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 TacoSwap (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 1st, 2021 – March 3rd, 2021.

Scope

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

Contract deployment address:
Repository
File:
TacoToken.sol
MasterChef.sol
Timelock.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 ▪ 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

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

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 well-secured.

Insecure	Poor secured	Secured	Well-secured

You are 

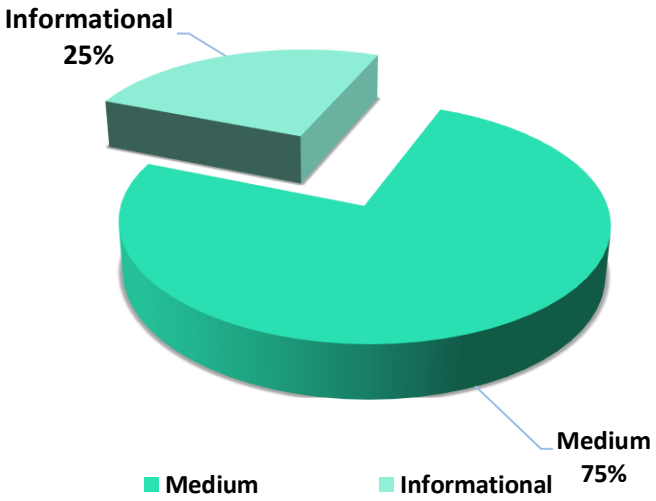
Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section,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

Security engineers found **2** medium, **1** informational issue during the audit.

Notice: the audit scope is limited and not include all files in the repository. Though, reviewed contracts are secure, we may not guarantee secureness of contracts that are not in the scope.

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

Timelock.sol

Description

Timelock queues and executes transactions.

Imports

Timelock has following imports:

- SafeMath.sol – from the OpenZeppelin.

Inheritance

Timelock does not inherit anything.

Usages

Timelock contract has following usages:

- SafeMath for uint.

Structs

Timelock contract has no data structures.

Enums

Timelock contract has no enums.

Events

Timelock contract has following events:

- event NewAdmin(address indexed newAdmin);
- event NewPendingAdmin(address indexed newPendingAdmin);
- event NewDelay(uint indexed newDelay);
- event CancelTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);
- event ExecuteTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);
- event QueueTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);

Modifiers
Timelock has no modifiers.

Fields

Timelock contract has following fields and constants:

- uint public constant GRACE_PERIOD = 14 days;
- uint public constant MINIMUM_DELAY = 6 hours;
- uint public constant MAXIMUM_DELAY = 30 days;
- address public admin – an admin address.
- address public pendingAdmin – a pending admin.
- uint public delay – delay between a transaction queueing and execution.
- mapping (bytes32 => bool) public queuedTransactions – queued transactions.

Functions
Timelock has following public functions:

- ***constructor***
  Description
  Inits the contract and sets default parameters.
  Visibility
  public
  Input parameters
  - address admin_ - admin address.
  - uint delay_ - delay between a transaction queuing and execution.

  Constraints
  - A `delay_` value should be between DELAY and MAXIMUM_DELAY.

  Events emit
  None
  Output
  None
- ***receive***
  Description
  Allows ETH transfers.
- ***setDelay***
  Description
  Sets a delay.
  Visibility
  public

Input parameters

- uint delay_ - delay between a transaction queuing and execution.

Constraints

- A message sender should be the contract itself.
- A `delay_` value should be between DELAY and MAXIMUM_DELAY.

Events emit

Emits the `NewDelay` event.

Output

None

- ***acceptAdmin***

Description

Accept the admin permissions.

Visibility

public

Input parameters

None

Constraints

- A message sender should be a pending admin.

Events emit

Emits the `NewAdmin` event.

Output

None

- ***setPendingAdmin***

Description

Accept the admin permissions.

Visibility

public

Input parameters

- address pendingAdmin_ - a pending admin address.

Constraints

- A message sender should be the contract itself.

Events emit

Emits the `NewPendingAdmin` event.

Output

None

- ***queueTransaction***

Description

Add a new transaction to the queue.

Visibility

public

Input parameters

- address target – a tx target.
- uint value – a tx value.
- string memory signature – a method signature.
- bytes memory data – a tx data.
- uint eta – a minimum delay between a tx queuing and execution.

Constraints

- A message sender should be admin.
- `eta` should be more than current time plus delay value.

Events emit

Emits the `QueueTransaction` event.

Output

bytes32 – a tx hash.

- ***cancelTransaction***

Description

Cancel a transaction.

Visibility

public

Input parameters

- address target – a tx target.
- uint value – a tx value.
- string memory signature – a method signature.
- bytes memory data – a tx data.
- uint eta – a minimum delay between a tx queuing and execution.

Constraints

- A message sender should be admin.

Events emit

Emits the `CancelTransaction` event.

Output

None

- ***executeTransaction***

Description

Execute a transaction.

Visibility

public

Input parameters

- address target – a tx target.
- uint value – a tx value.
- string memory signature – a method signature.
- bytes memory data – a tx data.

- uint eta – a minimum delay between a tx queuing and execution.

Constraints

- A message sender should be admin.
- A transaction should be queued.
- Current timestamp should be between `eta` and `eta` + GRACE_PERIOD.

Events emit

Emits the `ExecuteTransaction` event.

Output

None

MasterChef.sol

Description

MasterChef is a liquidity pool with rewards in Taco token.

Imports

MasterChef has following imports:

- @openzeppelin/contracts/math/SafeMath.sol
- ./libs/IBEP20.sol
- ./libs/SafeBEP20.sol
- @openzeppelin/contracts/access/Ownable.sol
- ./TacoToken.sol

Inheritance

MasterChef is Ownable.

Usages

MasterChef contract has following usages:

- SafeMath for uint256
- SafeBEP20 for IBEP20

Structs

MasterChef contract has following data structures:

- UserInfo
- PoolInfo

Enums

MasterChef contract has no enums.

Events

MasterChef contract has following events:

- Deposit
- Withdraw
- EmergencyWithdraw

Modifiers

MasterChef has no custom modifiers.

Fields

MasterChef contract has following fields and constants:

- TacoToken public taco
- address public devaddr
- uint256 public tacoPerBlock
- uint256 public constant BONUS_MULTIPLIER = 1
- address public feeAddress
- PoolInfo[] public poolInfo
- mapping (uint256 => mapping (address => UserInfo)) public userInfo
- uint256 public totalAllocPoint = 0
- uint256 public startBlock

Functions

MasterChef has following public functions:

- ***constructor***
  Description
  Sets initial values of the contract.
  Visibility
  public
  Input parameters
  - TacoToken _taco,
  - address _devaddr
  - address _feeAddress
  - uint256 _tacoPerBlock
  - uint256 _startBlock

Constraints
None
Events emit
None
Output
None

- ***poolLength***
  Description
  Returns a number of pools.
  Visibility
  external view
  Input parameters
  None
  Constraints
  None
  Events emit
  None
  Output
  - uint256 – a number of pools.
- ***changeFactor***
  Description
  Updates the *rewardTimeFactor*.
  Visibility
  public
  Input parameters
  None
  Constraints
  - onlyOwner modifier.

  Events emit
  None
  Output
  None
- ***add***
  Description
  Add a new lp to the pool.
  Visibility
  public
  Input parameters
  - uint256 _allocPoint
  - IERC20 _lpToken

- uint16 _depositFeeBP
- bool _withUpdate

Constraints

- onlyOwner modifier.

Events emit

None

Output

None

- ***set***

  Description

  Update the given pool's allocation point

  Visibility

  public

  Input parameters

  - uint256 _pid
  - uint256 _allocPoint
  - bool _withUpdate

  Constraints

  - onlyOwner modifier.

  Events emit

  None

  Output

  None

- ***getMultiplier***

  Description

  Return reward multiplier over the given _from to _to block.

  Visibility

  Public view

  Input parameters

  - uint256 from
  - uint256 to

  Constraints

  None

  Events emit

  None

  Output

  - uint256 – requested multiplier.

- ***pendingTaco***

  Description

  Returns pending reward tokens of a _user for a _pid reward pool.

Visibility
external view
Input parameters
- uint256 _pid
- address _user

Constraints
None
Events emit
None
Output
- uint256 – available tokens.

- ***massUpdatePools***
Description
Update reward variables for all pools.
Visibility
public
Input parameters
None
Constraints
None
Events emit
None
Output
None
- ***updatePool***
Description
Update reward variables of the given pool to be up-to-date.
Visibility
public
Input parameters
- uint256 _pid

Constraints
None
Events emit
None
Output
None
- ***deposit***
Description
Deposit LP tokens.

Visibility
public
Input parameters
- uint256 _pid
- uint256 _amount

Constraints
None
Events emit
Emits the Deposit event.
Output
None

- ***withdraw***

Description
Withdraw LP tokens.
Visibility
public
Input parameters
- uint256 _pid
- uint256 _amount

Constraints
- An _amount should not exceed a user balance of a _pid pool

Events emit
Emits the Withdraw event.
Output
None

- ***emergencyWithdraw***

Description
Withdraw LP tokens without a reward.
Visibility
public
Input parameters
- uint256 _pid

Constraints
None
Events emit
Emits the EmergencyWithdraw event.
Output
None

- ***dev***

Description

Allows dev address to set another dev address.

- ***setFeeAddress***
  Description
  Allows fee address to set another fee address.
- ***updateEmissionRate***
  Description
  Mass update pool and sets new tacoPerBlock value.
  Visibility
  public
  Input parameters
  - uint256 _tacoPerBlock

  Constraints
  - onlyOwner modifier.

  Events emit
  None
  Output
  None

TacoToken.sol

Description

TacoToken is a token with following parameters:

- Name: Taco
- Symbol: TACO
- Decimals: 18

The TacoToken has voting functionality.

Imports

TacoToken contract has following imports:

- ./libs/BEP20.sol

Inheritance

TacoToken contract is BEP20.

Usages

TacoToken contract has no custom usages.

Structs

TacoToken contract has following data structures:

- struct Checkpoint – stores votes checkpoints.

Enums

TacoToken contract has no custom enums.

Events

TacoToken contract has following custom evets:

- event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate)
- event DelegateVotesChanged(address indexed delegate, uint256 previousBalance, uint256 newBalance)

Modifiers

TacoToken has no custom modifiers.

Fields

TacoToken contract has following fields and constants:

- mapping (address => address) internal _delegates
- mapping (address => mapping (uint32 => Checkpoint)) public checkpoints
- mapping (address => uint32) public numCheckpoints
- bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)")
- bytes32 public constant DELEGATION_TYPEHASH = keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)")
- mapping (address => uint) public nonces

Functions

TacoToken has following public functions:

- ***delegates***
  Description
  Returns an address to whom *delegator* delegates his votes.
  Visibility

external view

Input parameters

- address delegator

Constraints

None

Events emit

None

Output

- address

- ***delegate***

  Description

  Delegate votes from *msg.sender* to *delegate*.

  Visibility

  external

  Input parameters

  - address delegatee

  Constraints

  None

  Events emit

  Emits *DelegateChanged* event.

  Output

  None

- ***delegateBySig***

  Description

  Delegates votes from signatory to *delegatee*.

  Visibility

  public

  Input parameters

  - address delegate
  - uint256 nonce
  - uint256 expiry
  - uint8 v
  - bytes32 r
  - bytes32 s

  Constraints

  None

  Events emit

  Emits *DelegateChanged* event.

  Output

  None

- ***getCurrentVotes***
  Description
  Get current votes balance for *account*.
  Visibility
  external view
  Input parameters
  - address account

  Constraints
  None
  Events emit
  None
  Output
  - uint256 — number of current votes for *account*.

- ***getPriorVotes***
  Description
  Determine the prior number of votes for an *account* as of a *blockNumber*.
  Visibility
  public view
  Input parameters
  - address account
  - uint256 blockNumber

  Constraints
  None
  Events emit
  None
  Output
  - uint256 — number of votes the account had as of the given block.

- ***mint***
  Description
  Mints an *_amount* to *_to* address.
  Visibility
  public
  Input parameters
  - address _to
  - uint256 _amount

  Constraints
  - *onlyOwner* modifier.

  Events emit

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

None

Output

None

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

Audit overview

Critical

No critical issues were found.

High

No high severity issues were found.

Medium

1. The *add* function of the *MasterChef* contract is lack of validations for the *_lpToken* existence.
2. The *updateEmissionRate* function of the *MasterChef* can fail due to block gas limit if the pool size is big enough.

Low

No low severity issues were found.

Lowest / Code style / Best Practice

1. Some code style issues were found by the static code analyzers.

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** medium, **1** informational issue during the audit.

Notice: the audit scope is limited and not include all files in the repository. Though, reviewed contracts are secure, we may not guarantee secureness of contracts that are not in the scope.

Violations in the following categories were found and addressed to Customer:

Category	Check Item	Comments
Code review	▪ Costly loops	▪ Execution of the updateEmissionRate function of the MasterChef may fail due to block gas limit
	▪ Data consistency	▪ The add function of the MasterChef is lack of _lpToken validation.

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.