HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Refinable
Date: April 4th, 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 are fixed - upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for Refinable.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Multiple purposes contracts
Platform	Ethereum, Binance Smart Chain, Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/refinableco/contracts-tge
Commit	
Deployed contract	
Timeline	28 MAR 2021 - 31 MAR 2021
Changelog	28 MAR 2021 - INITIAL AUDIT 4 APR 2021 - REMEDIATION CHECK

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 Refinable (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 28th, 2021 - March 31st, 2021.

Scope

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

```
Repository: https://github.com/refinableco/contracts-tge
Files:
      blockchain/contracts/interfaces/IBEP20.sol
      blockchain/contracts/libs/Context.sol
      blockchain/contracts/libs/Math.sol
      blockchain/contracts/libs/Ownable.sol
      blockchain/contracts/libs/SafeBEP20.sol
      blockchain/contracts/libs/SafeMath.sol
      blockchain/contracts/Factory.sol
      blockchain/contracts/Migrations.sol
      blockchain/contracts/RefinableToken.sol
      blockchain/contracts/TestCalls.sol
      blockchain/contracts/TimeLockedMultiSigWallet.sol
      blockchain/contracts/TimeLockedMultiSigWalletFactory.sol
      blockchain/contracts/TokenVesting.sol
      blockchain/contracts/TokenVestingFactory.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

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

	▪ 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

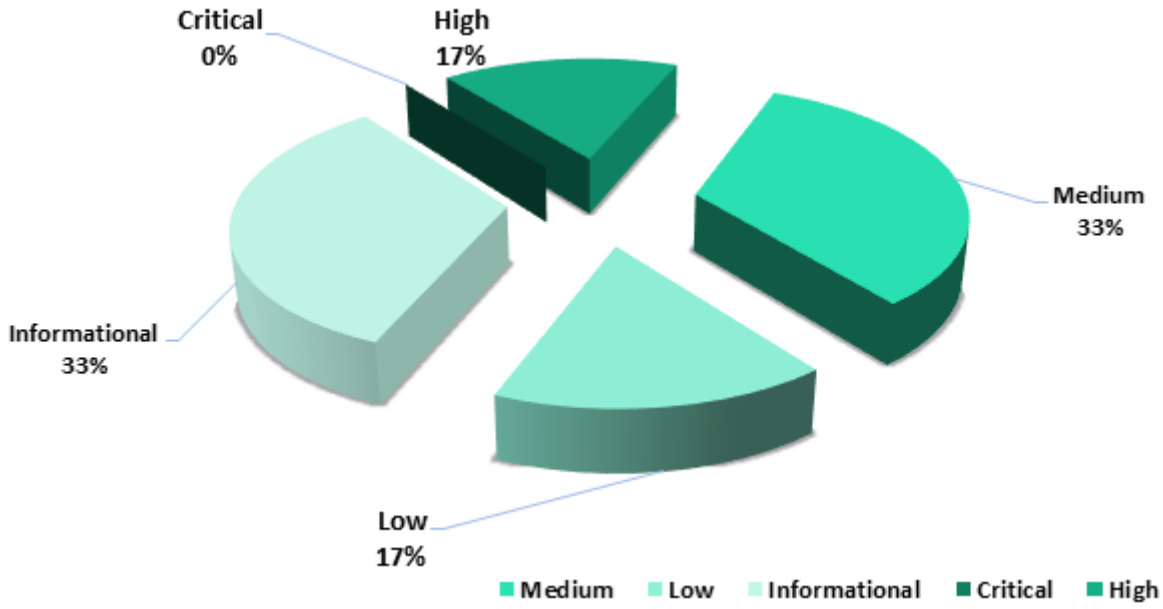
According to the assessment, the Customer's smart contracts are secure.

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 **0** critical, **1** high, **2** medium, **1** low, and **2** informational issues during the audit. All the issues were fixed for the secondary audit.

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.



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

Factory.sol

Description

Factory is a contract used to provide a framework for registering and generating contract factories.

Imports

Factory contract has no imports.

Inheritance

Factory contract does not inherit.

Usages

Factory contract has no usages.

Structs

Factory contract has no structs

Enums

Factory contract has no custom enums.

Events

Factory contract has following events:

- ContractInstantiation - Emitted when a contract is instantiated

Modifiers

Factory has no modifiers.

Fields

Factory contract has the following fields:

- mapping(address => bool) public isInstantiation - Used for tracing whether or not a contract has been instantiated
- mapping(address => address[]) public instantiations - Tracking the address of the instantiations

- mapping(address => string) public walletName - Tracking the character name of the wallet address that has been instantiated

Functions

Factory has no public functions.

RefineableToken.sol

Description

RefineableToken is a standard BEP-20 token (a representation of the ERC-20 standard utilized on the Binance Smart Chain). It has the following configuration supplied for its creation, and matches the standard BEP-20 token specification. The Refinable token has a fixed supply of 500 million tokens.

```
_name = 'Refinable Token';

_symbol = 'FINE';

_decimals = 18;

_totalSupply = 5 * 10**8 * 10**18; // 500m
```

TestCalls.sol

Description

TestCalls contains only those functions for usage such that one may test low-level calls issued from the multisig wallet. It should not be included within a production deployment.

TimeLockedMultiSigWalletFactory.sol

Description

TimeLockedMultiSigWalletFactory is a factory contract that allows for creation of numerous TimeLockedMultiSigWallet contracts. It possesses a single public function, create, which allows for a new TimeLockedMultiSigWallet to be generated, with variables _owners (the addresses of initial ownership), _required (Number of required confirmations), and _unlockDate (the date at which the multi-signature wallet is unlocked). The contract only allows one to generate a new TimeLockMultiSigWallet and not interact with existing TimeLockMultiSig wallets.

TimeLockedMultiSigWallet.sol

Description

TimeLockedMultiSigWallet provides a time-based vesting functionality.

Imports

TimeLockedMultiSigWallet contract has no imports

Inheritance

TimeLockedMultiSigWallet contract does not inherit from other contracts.

Usages

- *TimeLockedMultiSigWallet* contract has no usages.

Structs

TimeLockedMultiSigWallet contract has following data structures:

- Transaction - Used to store transaction information

Enums

TimeLockedMultiSigWallet contract has no enums

Events

TimeLockedMultiSigWallet contract has the following events:

- Confirmation - Emit when an owner confirms a transaction

- Revocation - Emit when an owner revokes a prior confirmation
- Submission - Emit when a new transaction is added to the transaction mapping
- Execution - Emit when a confirmed transaction is executed
- ExecutionFailure - Emit when a confirmed transaction fails to execute
- Deposit - Emit when a deposit is added
- OwnerAddition - Emit when an owner is added
- OwnerRemoval - Emit when an owner is removed
- RequirementChange - Emit when the number of required confirmation changes

Modifiers

TimeLockedMultiSigWallet has the following custom modifiers:

- onlyWallet - Sender is the wallet address
- ownerDoesNotExist - Used in cases in which the owner does not exist
- ownerExists - Used in cases in which the owner does exist
- TransactionExists - Used in cases in which the transaction exists
- confirmed - Used in cases in which an existing transaction is confirmed by owners
- notConfirmed - Used in cases in which an existing transaction is not yet confirmed
- notExecuted - Used in cases in which an existing transaction has been confirmed but not yet executed
- notNull - Used to ensure the address is not the 0 address
- isUnlocked - Used to validate whether or not the wallet is still in the timelock
- validRequirement - Used to ensure the owner count and number of required confirmation changes align

Fields

TimeLockedMultiSigWallet contract has following constants and fields:

- uint256 public constant MAX_OWNER_COUNT = 50;
- mapping(uint256 => Transaction) public transactions;
- mapping(uint256 => mapping(address => bool)) public confirmations;
- mapping(address => bool) public isOwner;

- address[] public owners;
- uint256 public required;
- uint256 public transactionCount;
- uint256 public unlockDate;

Functions

TimeLockedMultiSigWallet has following public functions:

- ***Fallback function***
  Description
  Allows transferring ETH to the contract.
- ***addOwner***
  Description
  Adds a new owner to the multi-sig wallet
  Visibility
  public
  Input parameters
  - address owner

  Constraints
  - New owner must not already exist
  - New owner must not be the 0 address
  - New owner must be within the acceptable amount of owners as determined by validRequirement

  Events emit
  Emits the OwnerAddition event.
  Output
  None
- ***removeOwner***
  Description
  Remove an owner from the multi-sig wallet
  Visibility
  public
  Input parameters
  - address owner

  Constraints
  - Only wallet can call

  Events emit
  Emits the OwnerRemoval event.
  Output
  None
- ***replaceOwner***
  Description

Replace an owner in the multi-sig wallet

Visibility

public

Input parameters

- address owner
- address newOwner

Constraints

- Only wallet can call
- owner must exist as owner
- newOwner must not exist as owner

Events emit

Emits the OwnerRemoval event and the OwnerAddition event.

Output

None

- ***changeRequirement***

Description

Change the number of required confirmations for the multi-signature wallet

Visibility

public

Input parameters

- uint256 _required

Constraints

- Only wallet can call
- Required number of confirmations must be a valid requirement length (as determined by the modifier)

Events emit

Emits the RequirementChange event

Output

None

- ***submitTransaction***

Description

Allows an owner to submit and confirm a transaction.

Visibility

public

Input parameters

- uint256 transactionId

Constraints

- N/A

Events emit

N/A

Output

None

- ***revokeConfirmation***
  Description
  Allows an owner to revoke a confirmation for a transaction.
  Visibility
  public
  Input parameters
    - uint256 transactionId

  Constraints
    - Owner must exist
    - Transaction must exist
    - Transaction must be confirmed
    - Transaction must not be already executed

  Events emit
  Emits the Revocation event
  Output
  None
- ***executeTransaction***
  Description
  Allows anyone to execute a confirmed transaction.
  Visibility
  public
  Input parameters
    - uint256 transactionId

  Constraints
    - Owner must exist
    - Transaction must exist
    - Transaction must be confirmed
    - Transaction must not be already executed

  Events emit
  Emits the Execution event in the event of a success, or the ExecutionFailure event in the event of a failed execution.
  Output
  None
- ***external_call***
  Description
  Creates a loop that copies tx.data into memory
  Visibility
  internal
  Input parameters
    - address destination
    - uint256 value

- uint256 dataLength
- bytes memory data

Constraints

- N/A

Events emit

N/A

Output

Result, which is the return data of the call

- ***getTransactionIds, getConfirmations, getTransaction, getUnlockDate, getOwners, getTransactionCount, getConfirmationCount***

  Description

  Simple view functions.

TokenVesting.sol

Description

TokenVesting is a token balance release contract which mirrors that of a traditional equity vest.

Imports

TokenVesting contract has the following imports:

- ../libs/SafeBEP20.sol
- ../libs/Ownable.sol
- ../libs/SafeMath.sol

Inheritance

TokenVesting contract is Ownable.

Usages

TokenVesting contract has following usages:

- SafeMath for uint256
- SafeBEP20 for IBEP20

Structs

TokenVesting contract has no structs.

Enums

TokenVesting contract has no enums.

Events

TokenVesting contract has no events.

Modifiers

TokenVesting has no modifiers.

Fields

TokenVesting contract has following constants and fields:

- address private _beneficiary - The beneficiary of the
- uint256 private _cliff -
- uint256 private _start -
- uint256 private _duration -
- bool private _revocable - A boolean for determining if a particular vesting contract can be revoked
- mapping(address => uint256) private _released - A mapping of which vesting contracts have been released
- mapping(address => bool) private _revoked - A mapping of which vesting contracts have been revoked

Functions

TokenVesting has following public functions:

- ***constructor***
  Description
  Initializes the contract.
  Visibility
  public
  Input parameters
  - address beneficiary - address of the beneficiary to whom vested tokens are transferred
  - uint256 start - the time (as Unix time) at which point vesting starts
  - uint256 cliffDuration - duration in seconds of the cliff in which tokens will begin to vest
  - uint256 duration - duration in seconds of the period in which the tokens will vest
  - bool revocable - whether the vesting is revocable or not

- address owner - the owner of the vesting contract

Constraints
None
Events emit
None
Output
None

- ***release***
  Description
  Transfers vested tokens to beneficiary.
  Visibility
  public
  Input parameters
  - IBEP20 token - The BEP20 token which is being vested

  Constraints
  None
  Events emit
  TokensReleased - Emit on the release of tokens to the beneficiary
  Output
  None
- ***revoke***
  Description
  Allows the owner to revoke the vesting. Tokens already vested remain in the contract, the rest are returned to the owner.
  Visibility
  public
  Input parameters
  - IBEP20 token - The BEP20 token which is being vested

  Constraints
  None
  Events emit
  TokenVestingRevoked - Emits following the revocation and transfer of the remaining non-vested tokens
  Output
  None
- ***beneficiary, cliff, start, duration, revocable, released, revoked, vestedAmount, _releaseableAmount, _vestedAmount***
  Description
  Simple view functions.

TokenVestingFactory.sol

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

Description

TokenVestingFactory is a factory contract that allows for creation of numerous TokenVesting contracts. It possesses a single public function, create, which allows for a new TokenVesting contract to be generated, with variables _beneficiary (the individual or entity who has the vesting interest), _start (time period at which vesting starts), _cliffDuration (the amount of time covered within the vesting cliff, the period during which the minimum cliff must first be reached prior to payout), _duration (the period of time during which the ownership is vesting), _revocable (a boolean which indicates whether or not the vesting can be revoked) and _owner (the ownership address who initiates the creation of the token vesting contract).

The contract only allows one to generate a new TokenVesting contract and not interact with existing TokenVesting contracts.

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

Audit overview

Critical

No critical findings

High

1. [Fixed] Hard-coded gas amounts in blockchain/contracts/TimeLockedMultiSigWallet.sol could lead to failed execution during periods of high gas volatility.

Medium

1. [Fixed] An older compiler version is used.

   We recommend updating to the latest stable one.

2. [Fixed] In the replaceOwner function in TimeLockedMultiSigWallet.sol, assert that the newOwner argument is not null to prevent assignment to a null address.

Low

1. [Fixed] The Gnosis MultiSig library used within the contracts has since been deprecated and usage of the SafeWallet (https://github.com/gnosis/safe-contracts) is now suggested.

Lowest / Code style / Best Practice

1. [Fixed] Multiple functions should be declared external in order to save gas.

2. [Fixed] Multiple code style issues were found by 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 the As-Is overview section of the report.

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

Security engineers found **0** critical, **1** high, **2** medium, **1** low, and **2** informational issues during the audit. All the issues were fixed for the secondary audit.

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

Category	Check Item	Comments
Code review	▪ Style Guide Violation	▪ Multiple instances of Mixed Case violation ▪ Multiple occurrences can be marked external (which does have gas savings)

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 platforms. 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.