HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: WOWToken
Date: February 24th, 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 WOWToken.
Type	ERC-20 token with specific functionality
Platform	Binance Smart Chain / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Approved by	Andrew Matiukhin | CTO and co-founder Hacken
BSCscan link Initial Audit	https://bscscan.com/address/0x1767102dc35b1593f39564a96aabbbd31fd302c2
BSCscan link Secondary Audit	https://bscscan.com/address/0x4da996c5fe84755c80e108cf96fe705174c5e36a https://bscscan.com/address/0x79f8ac4e7b4e83ca1ad4c54dfc5eaec659a1fe56
Timeline	18TH FEB 2021 – 24TH FEB 2021
Changelog	22ND FEB 2021 - Initial Audit 24TH FEB 2021 - Secondary Audit

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

Table of contents

Introduction 4
Scope 4
Executive Summary 5
Severity Definitions 6
AS-IS overview 7
Conclusion 17
Disclaimers 19

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

Introduction

Hacken OÜ (Consultant) was contracted by WOWToken (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 February 18th, 2021 – February 22nd, 2021. Secondary audit was done between February 23rd, 2021 – February 24th, 2021.

Scope

The scope of the project is main net smart contracts that can be found on BSCscan:
https://bscscan.com/address/0x4da996c5fe84755c80e108cf96fe705174c5e36a
https://bscscan.com/address/0x79f8ac4e7b4e83ca1ad4c54dfc5eaec659a1fe56
We have scanned this smart contract for commonly known and more specific vulnerabilities. List 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 ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

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

Executive Summary

According to the assessment, the Customer's smart contracts are well-secured.

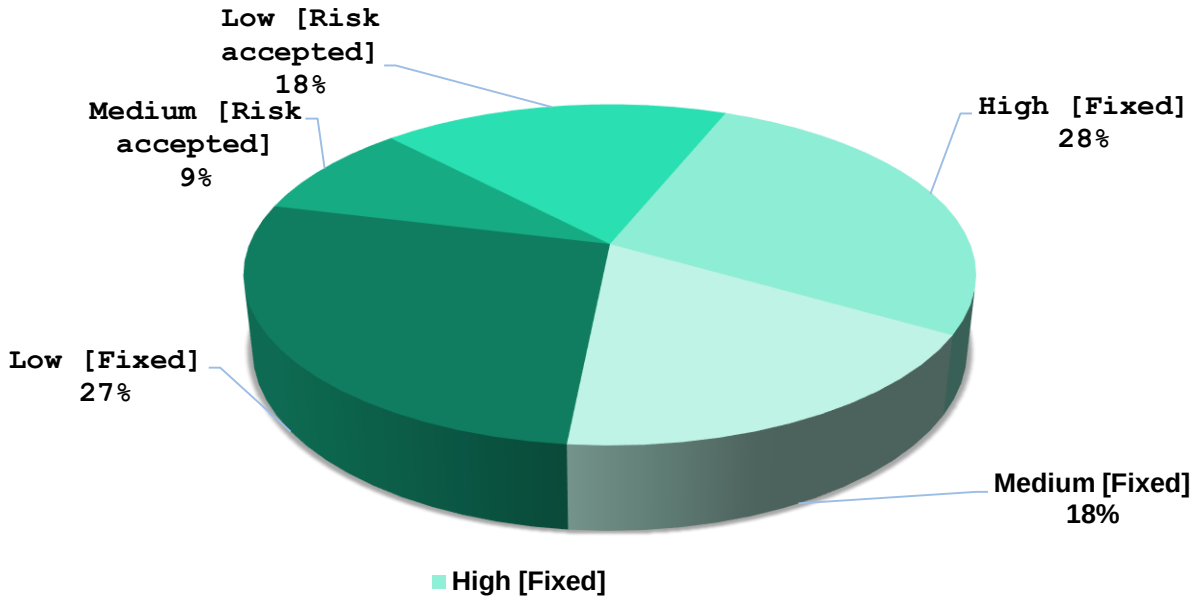
Insecure	Poor secured	Secured	Well-secured

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 3 high, 3 medium and 5 low severity issues during the initial audit. All this risks were mitigated or accepted by customer during secondary audit.

Graph 1. The distribution of vulnerabilities.



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 lose or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have 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 lose or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have 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

WOWToken smart contracts

WOWToken smart contract consists of library SafeMath, contract owned, contract ERC20Interface, contract ApproveAndCallFallBack, contract WOWSale, contract WOWToken.

SafeMath

Description

SafeMath is a standard OpenZeppelin library for mathematical operations to prevent overflows.

owned

Description

owned is a standard smart contract for basic access control with an owner role.

ERC20Interface

Description

ERC20Interface is a standard interface that describes functions for ERC20 token.

ApproveAndCallFallBack

Description

ApproveAndCallFallBack is a standard interface that describes fallback function on approve.

WOWSale

Description

WOWSale is a smart contract for WOWToken sale.

Imports

WOWSale is audited on-chain, thus, all imports are described above.

Inheritance

WOWSale contract is owned.

Usages

WOWSale contract has following usages:

- using SafeMath for uint256;

Structs

WOWSale contract has no custom structs.

Enums

WOWSale contract has no custom enums.

Events

WOWSale contract has following events:

- event ChangeRate(uint256 newRateUSD);
- event Sold(address buyer, uint256 amount);
- event CloseSale();

Modifiers

WOWSale contract has no custom modifiers.

Fields

WOWSale contract has following parameters:

- WOWToken public token;
- uint256 public totalSold;
- uint256 public rate = 200 ether;
- uint256 public tokenPrice = 5 ether;
- uint256 public startSale = 1614243600;
- uint256 public endSale = 1614502800;
- uint256 public maxBNB = 50 ether;
- uint256 public DEC;

Functions

WOWSale has following functions:

- ***constructor***
  Description

Sets WOWToken address
Visibility
public
Input parameters
- address wowToken

Constraints
None
Events emit
None
Output
None

- ***changeRate***
Description
changes USD to BNB rate
Visibility
public
Input parameters
- uint newRateUSD

Constraints
- onlyOwner

Events emit
- emit ChangeRate(newRateUSD);

Output
True

- ***changeEndSale***
Description
changes end sale date
Visibility
public
Input parameters
- uint256 newEndSale

Constraints
- onlyOwner

Events emit
None
Output
true

- ***buyTokens***
Description
performs token purchase
Visibility
public payable

Input parameters
None
Constraints
None
Events emit
- emit Sold(msg.sender, buyAmount);

Output
True

- ***close***
Description
Distributes collected BNB and returns unsold tokens
Visibility
public
Input parameters
None
Constraints
- onlyOwner

Events emit
- emit CloseSale();

Output
true

- ***fallback***
Description
calls buyTokens function
Visibility
public payable
Input parameters
None
Constraints
None
Events emit
None
Output
None

WOWToken

Description

WOWToken is a smart contract for ERC20 token.

Imports

WOWToken is audited on-chain, thus, all imports are described above.

Inheritance

WOWToken contract is ERC20Interface, owned.

Usages

WOWToken contract has following usages:

- using SafeMath for uint256;

Structs

WOWToken contract has no custom structs.

Enums

WOWToken contract has no custom enums.

Events

WOWToken contract has no custom events.

Modifiers

WOWToken contract has no custom modifiers.

Fields

WOWToken contract has following parameters:

- string public symbol = "WOW";
- string public name = "WOWswap";
- uint8 public decimals = 18;
- uint256 DEC = 10 ** uint256(decimals);
- uint256 public _totalSupply = 1000000 * DEC;
- mapping(address => uint) balances;
- mapping(address => mapping(address => uint)) allowed;

Functions

WOWToken has following functions:

- ***constructor***
  Description
  mints total supply to deployer
  Visibility
  public

Input parameters
None
Constraints
None
Events emit
- emit Transfer(0x0, owner, _totalSupply);

Output
None

- ***totalSupply***
Description
returns totalSupply
Visibility
public view
Input parameters
None
Constraints
None
Events emit
Name
Output
_totalSupply
- ***balanceOf***
Description
returns balance for address
Visibility
public view
Input parameters
None
Constraints
None
Events emit
None
Output
balances[tokenOwner]
- ***transfer***
Description
performs token transfer
Visibility
public
Input parameters
  - address to
  - uint tokens

Constraints
None
Events emit
- emit Transfer(msg.sender, to, tokens);

Output
true

- ***approve***

Description
performs token approve
Visibility
public
Input parameters
- address spender
- uint tokens

Constraints
None
Events emit
- emit Approval(msg.sender, spender, tokens);

Output
True

- ***increaseAllowance***

Description
Increases allowance for spender
Visibility
public
Input parameters
- address spender
- uint amount

Constraints
None
Events emit
None
Output
approve(spender, allowed[msg.sender][spender].add(amount));

- ***decreaseAllowance***

Description
Decreases allowance for spender
Visibility
public
Input parameters
- address spender

- uint amount

Constraints
None
Events emit
None
Output
approve(spender, allowed[msg.sender][spender].sub(amount));

- ***transferFrom***

Description
performs token transfer for approved spender
Visibility
public
Input parameters

- address from
- address to
- uint tokens

Constraints
None
Events emit

- emit Transfer(from, to, tokens);

Output
true

- ***allowance***

Description
returns allowance for owner and spender
Visibility
public view
Input parameters

- address tokenOwner
- address spender

Constraints
None
Events emit
Name
Output
allowed[tokenOwner][spender]

- ***approveAndCall***

Description
performs token approve with callback
Visibility
public
Input parameters

- address spender
- uint tokens
- bytes memory data

Constraints
None
Events emit
None
Output
true

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

Audit overview

Critical

No critical issues were found.

High

1. [Fixed] Approve function can't be used to change approved amount that is not 0. For example, if address1 approved 1 token to address to, it can't change approved amount to 2. It needs to change it to 0 and after that change to 2. It doesn't follow standard and may cause integration issues with other smart contracts.
2. [Fixed] WOWToken doesn't implement increaseAllowance and decreaseAllowance functions that protect from approve front running attacks. Absence of these functions may cause integration issues with other smart contracts.
3. [Fixed] The actual token rate for token sale will be different for most of configurations. For example, BNB price is set to 128 USD and token price is set to 5 USD, however, you'll get 25 tokens for 1 BNB because the integer division will remove remainder of the division.

Medium

4. [Fixed] SafeMath is used as contract not as library. It will cause additional gas usage for smart contract function calls.
5. [Fixed] It's highly recommended to split tokensale and token contract into 2 contracts so it will reduce potential risks for token after token sale is ended.

6. [Risk accepted] Because the price is set in USD it's possible to frontrun rate change and buy the token with lower rate. Generally, it's recommended to fix the sale rate in BNB and not to convert to USD.

Low

7. [Fixed] ERC20Interface doesn't implement standard ERC20 interface, it has additional non-ERC20 functions in it.
8. [Fixed] Solidity version is used is outdated, it has some known compiler issues for it. It's recommended to use higher compiler version.
9. [Fixed] Smart contract imports outdated libraries which may use additional gas on transactions.
10. [Risk accepted] Code is not covered with in-code documentations; it's recommended to add function description for all functions.
11. [Risk accepted] No unit tests were developed for the project. It's recommended to have 100% test coverage for code.

Lowest / Code style / Best Practice

No lowest issues were found.

Conclusion

Smart contracts within the scope was 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 3 high, 3 medium and 5 low severity issues during the initial audit. All this risks were mitigated or accepted by customer during secondary audit.

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 own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.