HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Bobo
Date: June 1st, 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 Bobo - Initial Audit
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC20 / Staking
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Zip archive	smart-contract-master.zip (md5: d92a224e4d9e24d3e430455b5cf59014)
Timeline	31 MAY 2021 - 1 JUNE 2021
Changelog	31 MAY 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 Bobo (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 on May 31th, 2021 - June 1st, 2021.

Scope

The scope of the project is the smart contract provided in zip archive:

```
smart-contract-master.zip (md5: d92a224e4d9e24d3e430455b5cf59014)
/contracts/Bobo.sol
/contracts/Staking.sol
/contracts/libraries/IterableSet.sol
/contracts/libraries/PriceOracle.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

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 ▪ Asset’s integrity ▪ User Balances 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 secured but could have more gas optimizations

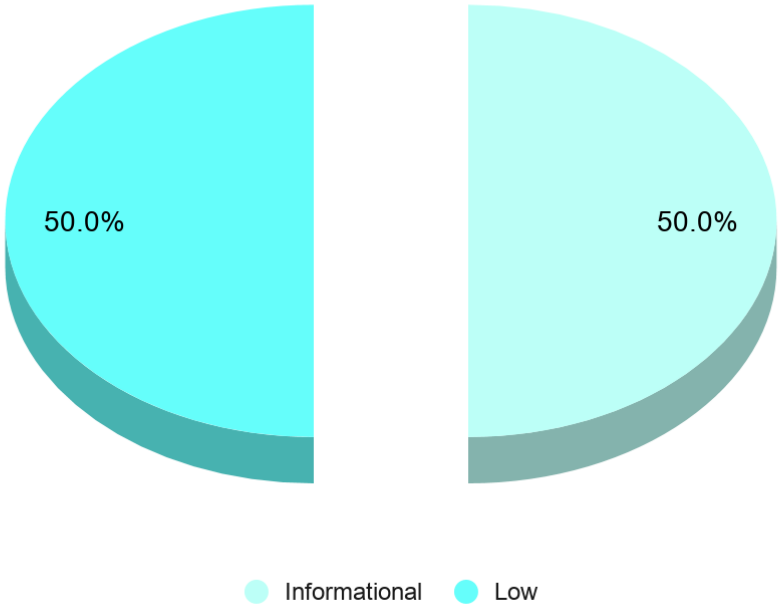
Insecure	Poor secured	Secured	Well-secured

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. All found issues can be found in the Audit overview section.

Security engineers found 3 low and 3 informational issues during the first review.

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

Audit overview

Critical

No Critical severity issues were found.

High

No High severity issues were found.

Medium

No Medium severity issues were found.

Low

1. Vulnerability: Pragma version not specified

   Source file does not specify required compiler version

   Recommendation: Consider adding "pragma solidity ^0.6.6;"

   Files:
   - libraries/PriceOracle.sol
   - libraries/IterableSet.sol

2. Vulnerability: Unused function parameter

   The function declares parameter which is never used

   Recommendation: Remove or comment out the variable name

   Lines: Bobo.sol#389-400

```
function fulfillRandomness(bytes32 requestId, uint256 randomness)
internal override {
   uint256 winnerIndex = _getIndex(randomness,
_holdersInLottery.length());
   address winner = _holdersInLottery.get(winnerIndex);

   uint amount = _lotteryTotal;
   _lotteryTotal = 0;

   _balances[address(this)] = _balances[address(this)].sub(amount);
   _balances[winner] = _balances[winner].add(amount);
   blockTimestampLast = block.timestamp;
   emit Winner(winner, amount);
}
```

3. Vulnerability: Function could be view

   Function state mutability can be restricted to view to save gas

   Recommendation: Please consider declaring function as view

   Lines: Bobo.sol#272-275

```
function _hasLiquidity() private returns (bool) {
   (uint112 reserve0, uint112 reserve1,) =
IUniswapV2Pair(address(uniswapV2Pair)).getReserves();
   return reserve0 > 0 && reserve1 > 0;
}
```

Lowest / Code style / Best Practice

1. Vulnerability: State variable should be declared as constant

   State variables that never change their values should be declared constants to save gas. Also, constants should be named UPPER_CASE_WITH_UNDERSCORES

   Recommendation: Please consider declaring unchanged variables as constants

   Lines: Bobo.sol#34-37

```
uint256 private _initialSupply = 8888888 * 10 ** 18;
uint256 private _lpFarmingSupply = 880000000 * 10 ** 18;
uint256 private _initialLimitForTransfer = 888888 * 10 ** 16;
uint256 private _totalSupply = _lpFarmingSupply + _initialSupply;
```

   Lines: Bobo.sol#53

```
uint256 private _minForLottery = 1 * 10 ** 18;
```

   Lines: Bobo.sol#42-44

```
string private _name = "Bobo Carpet";
string private _symbol = "BOBO";
uint8 private _decimals = 18;
```

   Lines: Bobo.sol#61-65

```
address private addressLink =
0x404460C6A5EdE2D891e8297795264fDe62ADBB75;
address private addressVRF =
0x747973a5A2a4Ae1D3a8fDF5479f1514F65Db9C31;
address private addressRouterV2 =
0x05fF2B0DB69458A0750badebc4f9e13aDd608C7F;
```

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

```
bytes32 internal keyHash =
0xc251acd21ec4fb7f31bb8868288bfdbaeb4fbfec2df3735ddbd4f7dc8d60103c;
uint256 internal fee = 0.2 * 10 ** 18;
```

Lines: Bobo.sol#58

```
uint256 private numTokensSellToAddToLiquidity = 500000 * 10 ** 18;
```

2. Vulnerability: Public function that could be declared external

   public functions that are never called by the contract should be declared external to save gas.

Lines: Bobo.sol#91

```
function name() public view returns (string memory) {
```

Lines: Bobo.sol#95

```
function getCurrentLiquidity() public view returns (uint256) {
```

Lines: Bobo.sol#99

```
function getCurrentLottery() public view returns (uint256) {
```

Lines: Bobo.sol#103

```
function getLotteryParticipantByIndex(uint256 index) public view
returns (address) {
```

Lines: Bobo.sol#107

```
function getLotteryParticipantLength() public view returns (uint256) {
```

Lines: Bobo.sol#111

```
function symbol() public view returns (string memory) {
```

Lines: Bobo.sol#115

```
function decimals() public view returns (uint8) {
```

Lines: Bobo.sol#119

```
function totalSupply() public view override returns (uint256) {
```

Lines: Bobo.sol#123

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

```
function balanceOf(address account) public view override returns
(uint256) {
```

Lines: Bobo.sol#127

```
function transfer(address recipient, uint256 amount) public override
returns (bool) {
```

Lines: Bobo.sol#132

```
function allowance(address owner, address spender) public view override
returns (uint256) {
```

Lines: Bobo.sol#136

```
function approve(address spender, uint256 amount) public override
returns (bool) {
```

Lines: Bobo.sol#141

```
function transferFrom(address sender, address recipient, uint256
amount) public override returns (bool) {
```

Lines: Bobo.sol#147

```
function increaseAllowance(address spender, uint256 addedValue) public
virtual returns (bool) {
```

Lines: Bobo.sol#152

```
function decreaseAllowance(address spender, uint256 subtractedValue)
public virtual returns (bool) {
```

Lines: Bobo.sol#183

```
function isExcludedFromFee(address account) public view returns (bool)
{
```

Lines: Bobo.sol#337

```
function getInfo() public view returns (uint priceCumulative, uint
circulatingSupply, uint marketCap) {
```

Lines: Bobo.sol#350

```
function getStakingAddress() public view returns (address) {
```

Lines: Bobo.sol#354

```
function getPairAddress() public view returns (address) {
```

Lines: Staking.sol#45

```
function deposit(uint _amount) public nonReentrant {
```

Lines: Staking.sol#64

```
function withdraw() public nonReentrant {
```

Lines: Staking.sol#85

```
function claim() public nonReentrant {
```

Lines: Staking.sol#101-105

```
function availableForClaim(address account) public view returns (
    uint reward,
    uint lastAccumulatedBlock,
    uint latestBlock
) {
```

Lines: Staking.sol#136

```
function getDepositedBalance(address account) public view returns
(uint) {
```

Lines: Staking.sol#141

```
function getLpSupply() public view returns (uint) {
```

3. The length of lines 141, 152, 286, 295, 320, 337 and 344 of Bobo.sol and 52, 71, 92, 93 and 111 of Staking.sol exceeds recommended maximum line length

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.

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

Security engineers found 3 low and 3 informational issues during the first review.

Category	Check Items	Comments
➔ Code Review	➔ Style guide violation	➔ Maximum line length
	➔ Gas Savings	➔ public function should be external ➔ state variable should be constant ➔ Unused function parameter ➔ function should be view

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 the 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.