HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Buoy

Date: October 22nd, 2020

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 Buoy.
Type	ERC-20 token with buying functionality.
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Approved by	Andrew Matiukhin | CTO and co-founder Hacken
Repository URL	https://github.com/AlanStacks/Buoy
Commit	344089E1DBDCAE1CE122D1F5CBC1A66A1BD789EA
Timeline	19TH OCT 2020 – 22ND OCT 2020
Changelog	22ND OCT 2020 - 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 Buoy (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 October 19th, 2020 – October 22nd, 2020.

Second audit conducted between November 6, 2020 – November 8, 2020.

Addresses verification conducted on November 18, 2020.

Scope

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

Repository URL - https://github.com/AlanStacks/Buoy
Commit - 344089E1DBDCAE1CE122D1F5CBC1A66A1BD789EA

Contract	In scope	Deployed contract address
Buoy.sol	Yes	0xa6444f5234e9C1C688a8EB71273376971bE48456
DavyJones.sol	Yes	0xaBa513097f04D637727FDCda0246636E0D5D6833
BPool.sol	No	0x2bb0ceed9083ab2bb1ff5270608d642835ddf864

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
Functional review	▪ Business Logics Review ▪ Functionality Checks

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

	▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

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

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 3 critical, 3 medium and 2 low severity issues during the audit. The code is not tested and not following project structure best practices.

After the second audit, security engineers found additional issues: 1 critical. The code is not tested.

After the address's verification process, we confirm that a poolAddress is set up to the BalancerPool (BPool) address and the corresponding critical issue can be considered as fixed.

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

Buoy.sol

Description

Buoy is an ERC-20 token with sale functionality. Allows to buy tokens for the limited period.

Imports

Buoy contract has no custom imports.

Inheritance

Buoy contract is Owned.

Usages

Buoy contract uses:

- *using SafeMath for uint256;*

Structs

Buoy contract has no custom data structures.

Enums

Buoy contract has no custom enums.

Events

Buoy contract has no custom events.

Modifiers

Buoy contract has no custom modifiers.

Fields

Buoy contract has following fields and constants:

- *mapping (address => uint256) private _balances;* - token balances.
- *mapping (address => mapping (address => uint256)) private _allowances;* - allowances.
- *mapping (address => uint) reserves;* - sale reserves.
- *mapping (address => uint) ethPaid;* - ETH paid in sail period.

- *bool withdrawable;* - indicates whether sold tokens are withdrawable.
- *bool withdrawPeriodOver;* - end of the withdraw period.
- *bool poolMinted;* - indicates whether an initial pool is minted.
- *bool addressLocked;* - indicates whether davysAddress is set.
- *bool ethInjected;* - indicates whether ETH is sent to the DavyJones contract.
- *bool saleHalted;* - indicates whether sale is halted.
- *string private _name = "Buoy";* - the token name.
- *string private _symbol = "BUOY";* - the token symbol.
- *uint8 private _decimals = 18;* - decimals of the token.
- *uint _totalSupply;* - total supply of the token.
- *uint _totalReserved;* - total reserves of the token. Reserves are used during the token sale.
- *uint startDate;* - token sale start date.
- *uint stage1;* - token sale first stage date.
- *uint stage2;* - token sale second stage date.
- *uint endDate;* - token sale end date.
- *uint safetySwitch;* - date when the `*publicInjectLiquidity*` function can be called.
- *uint withdrawalLimit;* - withdrawal date upper limit.
- *uint nonce = 0;* - current state of the sale.
- *address payable public davysAddress;* - the DaveJones contract address.
- *address payable public buoyPresale = 0xD10Fd220efC658E72fcB09a1422394eE48A39d54;* - the Buoy presale address.

Functions

Buoy has following functions:

- ***constructor***
  Description
  Initializes the contract. Sets token parameters, sets the contract owner and calls a `*_mintOriginPool*` function
  Visibility
  public
  Input parameters
  None
  Constraints
  None
  Events emit

None
Output
None

- ***totalSupply, balanceOf, allowance, transfer, approve, increaseAllowance, decreaseAllowance, burn***
  Description
  Default ERC-20 functions.
- ***balanceOfMe***
  Description
  Get token balance of the caller.
  Visibility
  Public
  Input parameters
  None
  Constraints
  None
  Events emit
  None
  Output
  - uint256 – token balance of the caller.
- ***burnFrom***
  Description
  Burn tokens from an *`address`*.
  Visibility
  External
  Input parameters
  - address account
  - uint256 amount

  Constraints
  Allowance should be set.
  Events emit
  Emits the *`Transfer`* event.
  Output
  - uint256 – token balance of the caller.
- ***startSale***
  Description
  Sets startDate to now and defines the sale stages.
  Visibility
  Public
  Input parameters

None

Constraints

- onlyOwner modified

Events emit

None

Output

None

- ***buySale***

Description

Buy tokens for ETH.

Visibility

Public

Input parameters

None

Constraints

- sale should be active
- one-time sum of bought tokens should be less or equal to 999960 tokens.

Events emit

None

Output

None

- ***fallback function ()***

Description

Allows the contract to receive ETH. All received ETH processed by the `*buySale*` function.

- ***withdraw***

Description

Withdraw tokens after the token sale.

Visibility

Public

Input parameters

None

Constraints

- a caller should have reserved tokens.
- withdrawals should be active.

Events emit

None

Output

None

- ***endWithdrawPeriod***
  Description
  Clears reserves.
  Visibility
  Public
  Input parameters
  None
  Constraints
  - onlyOwner modifier.
  - withdrawals period should be ended.

  Events emit
  None
  Output
  None
- ***redeemPresale***
  Description
  Exchange presale tokens to the reserved tokens.
  Visibility
  Public
  Input parameters
  None
  Constraints
  - sale should be active
  - allowance should be set

  Events emit
  None
  Output
  None
- ***viewStage, viewPossibleReserved, viewReserved, viewMyReserved, viewMyEthPaid, viewEthRaised***
  Description
  View functions used to receive corresponding sale information.
- ***setAddresses***
  Description
  Sets the DavyJones contract address.
  Visibility
  Public
  Input parameters
  - address payable davy

  Constraints

- onlyOwner modifier.
- The address should not be set yet.

Events emit
None
Output
None

- ***lockAddresses***

Description
Locks the DavyJones contract address.
Visibility
Public
Input parameters
None
Constraints
- onlyOwner modifier.
- The address should be set.

Events emit
None
Output
None

- ***injectLiquidity***

Description
Transfers ETH to the DavyJones contract, and mints liquidity tokens to it.
Visibility
Public
Input parameters
- uint gasPrice – gas limit for sending ETH.

Constraints
- onlyOwner modifier.
- Sale should be finished.

Events emit
None
Output
None

- ***haltSale***

Description
Stops sale and allows to refund ETH.
Visibility
Public
Input parameters

None

Constraints

- onlyOwner modifier.
- Sale should be active.

Events emit

None

Output

None

- ***emergencyRefund***

  Description

  Refund all invested ETH of a caller. Contains reentrancy vulnerability.

  Visibility

  Public

  Input parameters

  None

  Constraints

  - Sale should be halted.
  - A caller should have invested ETH.

  Events emit

  None

  Output

  - bytes memory

- ***endSaleEarly***

  Description

  End sale if maximum tokens cap is collected.

  Visibility

  Public

  Input parameters

  None

  Constraints

  - Maximum tokens cap is collected

  Events emit

  None

  Output

  None

- ***publicInjectLiquidity***

  Description

  Transfers ETH to the DavyJones contract, and mints liquidity tokens to it.

  Visibility

  Public

Input parameters

- None uint gasPrice – gas limit for sending ETH.

Constraints

- Sale should be finished.
- Liquidity should not be injected yet.
- safetySwitch date should be passed.

Events emit

None

Output

None

DavyJones.sol

Description

DavyJones accepts ETH and Buoy tokens and allows transfers to the pool and uniswap router.

Imports

DavyJones contract has no custom imports.

Inheritance

DavyJones contract is Owned.

Usages

DavyJones contract has no custom uses.

Structs

DavyJones contract has no custom data structures.

Enums

DavyJones contract has no custom enums.

Events

DavyJones contract has no custom events.

Modifiers

DavyJones contract has no custom modifiers.

Fields

DavyJones contract has following fields and constants:

- *uint approvalAmount = 999999999999 * (10 ** 18);*
- *uint safetyRelease = 999999999999;*
- *uint withdrawlCheck;*
- *uint256[] index = [approvalAmount,approvalAmount,approvalAmount,approvalAmount,approvalAmount,approvalAmount,approvalAmount,approvalAmount];*
- *address weth = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;*
- *address buidl = 0x7b123f53421b1bF8533339BFBdc7C98aA94163db;*
- *address dxd = 0xa1d65E8fB6e87b60FECCBc582F7f97804B725521;*
- *address bal = 0xba100000625a3754423978a60c9317c58a424e3D;*
- *address mkr = 0x9f8F72aA9304c8B593d555F12eF6589cC3A579A2;*
- *address lrc = 0xBBbbCA6A901c926F240b89EacB641d8Aec7AEafD;*
- *address link = 0x514910771AF9Ca656af840dff83E8264EcF986CA;*
- *address comp = 0xc00e94Cb662C3520282E6f5717214004A7f26888;*
- *address public buoy;*
- *address public pool;*
- *address public swap = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;*
- *address[] buidlPath = [weth,buidl];*
- *address[] dxdPath = [weth,dxd];*
- *address[] balPath = [weth,bal];*
- *address[] mkrPath = [weth,mkr];*
- *address[] lrcPath = [weth,lrc];*
- *address[] linkPath = [weth,link];*
- *address[] compPath = [weth,comp];*
- *address[] unbuidlPath = [buidl,weth];*
- *address[] undxdPath = [dxd,weth];*
- *address[] unbalPath = [bal,weth];*
- *address[] unmkrPath = [mkr,weth];*
- *address[] unlrcPath = [lrc,weth];*
- *address[] unlinkPath = [link,weth];*
- *address[] uncompPath = [comp,weth];*
- *bool addressLocked;*
- *bool liquidityBurnt;*
- *bool approved;*
- *SwapInterface swapContract = SwapInterface(swap);*

Functions

DavyJones has following functions:

- ***constructor***
  Description
  Sets the contract owner.
  Visibility
  public
  Input parameters
  None
  Constraints
  None
  Events emit
  None
  Output
  None
- ***setBuoyAndPoolAddress***
  Description
  Sets the Buoy and pool addresses.
  Visibility
  public
  Input parameters
  - address by
  - address pl

  Constraints
  - onlyOwner modifier.
  - addresses should not be set.

  Events emit
  None
  Output
  None
- ***lockAddress***
  Description
  Locks the Buoy and pool addresses.
  Visibility
  public
  Input parameters
  None
  Constraints
  - onlyOwner modifier.

- addresses should be set.

Events emit
None
Output
None

- ***deposit***
  Description
  Send tokens to the pool.
  Visibility
  public
  Input parameters
  - uint bpt

  Constraints
  - onlyOwner modifier.

  Events emit
  None
  Output
  None
- ***fallback function ()***
  Description
  Receives ETH and swap it to tokens.
  Visibility
  external
  Input parameters
  None
  Constraints
  - can only be called from the Buoy contract.
  - Addresses should be locked.

  Events emit
  None
  Output
  None
- ***unswapLeftovers***
  Description
  Send all tokens to the contract owner in a case when at least 98% of tokens are already spent by the contract.
  Visibility
  public
  Input parameters

None

Constraints

- 98% of tokens should already be spent.

Events emit

None

Output

None

- ***publicDeposit***

Description

Send tokens to the pool.

Visibility

public

Input parameters

- uint bpt

Constraints

- safetyRelease period should pass.

Events emit

None

Output

None

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

Audit overview

Critical

1. The *`buySale`* function is supposed to limit total amount of sold tokens to 1M but compares tokens that a user wants to buy with *`_totalSupply`* variable that is always equal to 40 tokens before sale is not finished. As a result, it is possible to buy an unlimited number of tokens.

   Fixed before the second audit.

2. The *`emergencyRefund`* function is vulnerable to the reentrancy attack. The simple code allowing to refund x3 of ETH provided here: https://www.codepile.net/pile/a2wK0p2n.

   We recommend to use send or transfer functions and to set ethPaid to 0 before sending ETH.

   Fixed before the second audit.

3. The *`emergencyRefund`* function allows to refund ETH but does not set reserved tokens to 0. In a case when the sale is started again, those who refund will get free tokens.

   Fixed before the second audit.

4. The contract with *`poolAddress`* is not provided and can not be verified. This address has access to all tokens swapped from the Uniswap.

   We recommend an additional verification of the *`poolAddress`* code after its set up on the deployed contract.

   The `poolAddress` points to the BalancePool (BPool) contract in the deployed version of the contract.

High

No high severity issues were found.

Medium

1. The *`Buoy`* has custom implementation of ERC-20 functionality. We recommend using OpenZeppelin version by importing it to the contract instead of writing own code.

   Fixed before the second audit.

2. The *`poolMinted`* variable is useless because the *`_mintOriginPool`* is called only from the *`constructor`* and is never used elsewhere. All the operations with this variable can be removed.

   Fixed before the second audit.

3. The *`gasPrice`* parameter of the *`injectLiquidity`* function is actually a gas limit and can be removed. We recommend sending ETH to simple wallets by calling `transfer` or `send` functions. In case when ETH is sent to a contract and requires some specific actions, we recommend to hardcode the gas limit.

   Fixed before the second audit.

Low

1. _approve… and _unswap… functions are duplicated. Those functions can merged and contract addresses could be passed to new functions as arguments.

   Fixed before the second audit.

2. All tokens addresses are hardcoded. That complicates testing of smart contracts.

   Fixed before the second audit.

3. *`_totalSupply`* variable is declared but never used.

   Fixed before the second audit.

Lowest / Code style / Best Practice

1. A lot of code style issues have been 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 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 critical, 3 medium and 2 low severity issues during the audit. The code is not tested and not following project structure best practices.

After the second audit, security engineers found additional issues: 1 critical issue. The code is not tested.

After the address's verification process, we confirm that a poolAddress is set up to the BalancerPool (BPool) address and the corresponding critical issue can be considered as fixed.

Violations in following categories were found and addressed to Customer:

Category	Check Item	Comments
Code review	▪ Repository consistency	▪ The project does not follow project structure best practices and lack of tests.

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.