HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Pulse
Date: February 2nd, 2021

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

Document

Name	Smart Contract Code Review and Security Analysis Report for Pulse.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Token, token sale, staking.
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	
Commit	
Deployed contract	Pulse: https://etherscan.io/address/0x7927a3ad11d02c73467f62a36bf3e06d6d6a3437 PulseSale: https://etherscan.io/address/0x0bfe90382e2dd094a4b65fdafb6cd2010fafdd86 PulseStake: https://etherscan.io/address/0xbfce60ec99d2fff5f3ceedd248545f458ef40d85
Timeline	01 FEB 2021 - 03 FEB 2021
Changelog	02 FEB 2021 - INITIAL AUDIT 03 FEB 2021 - SECOND REVIEW

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 Pulse (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 1st, 2021 - February 2nd, 2021.

Second review February 3rd, 2021

Scope

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

```
Contract deployment address:
Repository
File:
      Pulse.sol
      PulseSale.sol
      PulseStake.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 secure.

Insecure	Poor secured	Secured	Well-secured

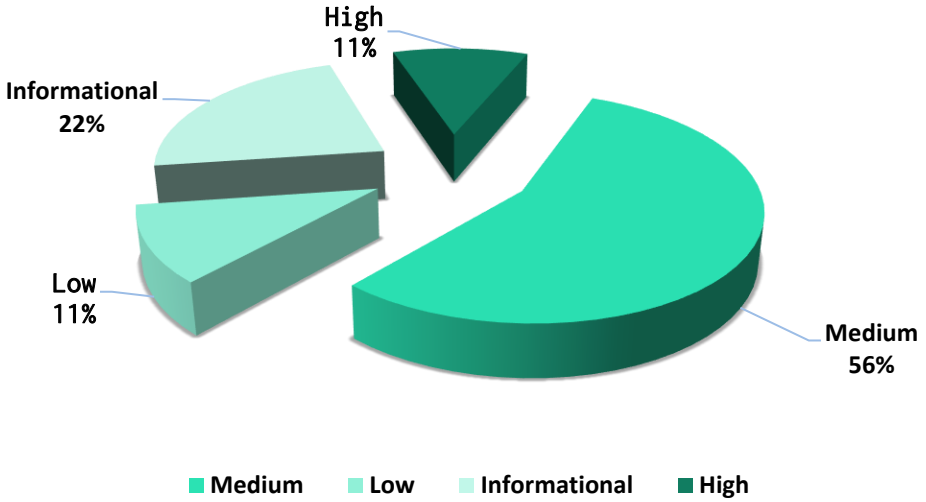
You are

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 1 high, 5 medium, 1 low and 2 informational issues during the audit.

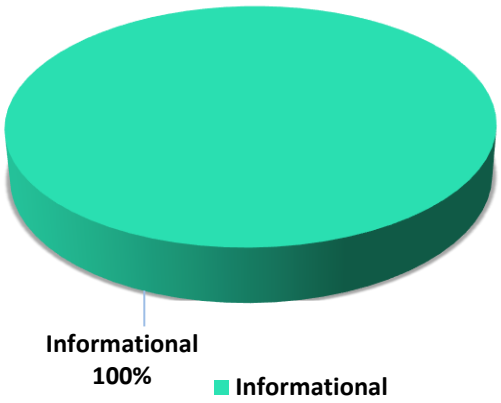
After the second review, Customers` smart contracts contains 2 informational issues.

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

Graph 2. The distribution of vulnerabilities after the second 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

Pulse.sol

Description

Pulse is an ERC-20 token with fees applied to each transfer. No fees are applied if a receiver or a sender are marked as `feeless`.

Token name: PULSEDEFI.LTD.

Token symbol: PULSE.

Tokens allocations: 1 million tokens to the message sender.

Imports

Pulse contract has the following imports:

- hardhat/console.sol
- @openzeppelin/contracts/access/Ownable.sol
- @openzeppelin/contracts/math/SafeMath.sol
- ./ERC20.sol

Inheritance

Pulse contract is ERC20, Ownable.

Usages

Pulse contract has following usages:

- SafeMath for uint256

Structs

CropToken contract has no custom data structures.

Enums

CropToken contract has no custom enums.

Events

CropToken contract has following events:

- feeReceiverChanged
- UpdatedFeelessAddress

Modifiers

CropToken has no custom modifiers.

Fields

CropToken contract has following custom fields and constants:

- uint32 public txFee
- uint32 public feeDivisor
- address public feeReceiver
- mapping(address => bool) public feeless

Functions

CropToken has following external or public functions:

- *constructor*
  Description
  Deploys the contract. Mints 1M tokens to the message sender.
  Visibility
  public
  Input parameters
  - uint32 _initFee
  - uint32 _initDivisor
  - address _initFeeReceiver

  Constraints
  None
  Events emit
  None
  Output
  None
- *burn*
  Description
  Burns an `amount` of tokens belongs to the message sender.
  Visibility
  public
  Input parameters
  - uint256 amount

  Constraints
  None
  Events emit
  None
  Output
  None
- *changeFee*
  Description
  Sets the `txFee` value.
  Visibility
  public
  Input parameters

    - uint32 _newTxFee

  Constraints
    - `onlyOwner` modifier

  Events emit
  None
  Output
  None

- *changeFeeReceiver*
  Description
  Sets the `feeReceiver` address.
  Visibility
  public
  Input parameters
    - address _receiver

  Constraints
    - `onlyOwner` modifier

  Events emit
  Emits the `feeReceiverChanged` event.
  Output
  None
- *updateFeelessAddress*
  Description
  Marks an `_address` as `_feeless`.
  Visibility
  public
  Input parameters
    - address _address
    - bool _feeless

  Constraints
    - `onlyOwner` modifier

  Events emit
  Emits the `UpdatedFeelessAddress` event.
  Output
  None

PulseSale.sol

Description

PulseSale is a token sale contract.

Imports

PulseSale contract has the following imports:

- hardhat/console.sol
- @openzeppelin/contracts/access/Ownable.sol
- @openzeppelin/contracts/GSN/Context.sol

- @openzeppelin/contracts/token/ERC20/IERC20.sol
- @openzeppelin/contracts/math/SafeMath.sol
- @openzeppelin/contracts/token/ERC20/SafeERC20.sol
- @openzeppelin/contracts/utils/ReentrancyGuard.sol
- @openzeppelin/contracts/utils/Pausable.sol

Inheritance

PulseSale contract is Context, ReentrancyGuard, Ownable, Pausable

Usages

PulseSale contract has following usages:

- SafeMath for uint256
- SafeERC20 for IERC20

Structs

PulseSale contract has no custom data structures.

Enums

PulseSale contract has no custom enums.

Events

PulseSale contract has following events:

- TokensPurchased
- AdminRescueTokens

Modifiers

PulseSale has no custom modifiers.

Fields

PulseSale contract has following custom fields and constants:

- IERC20 public Pulse
- address payable private _wallet
- uint256 private _rate
- uint256 private _weiRaised
- uint256 private _cap
- uint256 private constant DECIMALS = 1e18
- uint256 private _tokensSold
- bool public isClaimabl
- mapping(address => uint256) private _balances;

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

Functions

PulseSale has following external or public functions:

- *constructor*
  Description
  Deploys the contract.
  Visibility
  public
  Input parameters
  - uint256 rate
  - address payable wallet
  - IERC20 pulseToken
  - uint256 cap

  Constraints
  None
  Events emit
  None
  Output
  None
- *token, wallet, rate, weiRaised, tokensSold, tokensOwed*
  Description
  Simple view functions.
- *buyTokens, receive*
  Description
  Allows buying tokens in exchange of ETH.
  Visibility
  public
  Input parameters
  - address beneficiary

  Constraints
  - `nonReentrant` modifier.
  - `whenNotPaused` modifier.

  Events emit
  Emits the `TokensPurchased` event.
  Output
  None
- *claimTokens*
  Description
  Claims all purchased tokens.
  Visibility
  public
  Input parameters
  - address beneficiary

  Constraints
  - `nonReentrant` modifier.
  - `whenNotPaused` modifier.

- `isClaimable` should be set by the owner.
- A message sender should have claimable tokens.

Events emit
None
Output
None

- ***changeRate***
  Description
  Changes an exchange rate.
  Visibility
  external
  Input parameters
  - uint256 _weiRat

  Constraints
  - `onlyOwner` modifier.
  - `whenPaused` modifier.

  Events emit
  None
  Output
  None
- ***makeClaimable***
  Description
  Makes tokens claimable.
  Visibility
  external
  Input parameters
  None
  Constraints
  - `onlyOwner` modifier.

  Events emit
  None
  Output
  None
- ***adminRescueTokens***
  Description
  Allows to withdraw any tokens that are accidently sent to the contract.
  Visibility
  external
  Input parameters
  - address rescuedToken
  - address recipient
  - uint256 amount

  Constraints
  - `onlyOwner` modifier.

- `rescuedToken` and `recipient` should not be 0 addresses.
- `amount` should be grater than 0.

Events emit
Emits the `AdminRescueTokens` event.
Output
None

PulseStake.sol

Description

PulseStake is a contract that allows to stake tokens.

Imports

PulseSale contract has the following imports:

- hardhat/console.sol
- @openzeppelin/contracts/token/ERC20/IERC20.sol
- @openzeppelin/contracts/math/SafeMath.sol
- @openzeppelin/contracts/GSN/Context.sol
- @openzeppelin/contracts/access/Ownable.sol
- @openzeppelin/contracts/utils/ReentrancyGuard.sol

Inheritance

PulseSale contract is Ownable, ReentrancyGuard.

Usages

PulseSale contract has following usages:

- SafeMath for uint256

Structs

PulseSale contract has following data structures:

- UserStakeBracketInfo - stores a user stake info.

Enums

PulseSale contract has no custom enums.

Events

PulseSale contract has following events:

- userStaked

- userClaimed
- stakeRewardUpdated

Modifiers

PulseSale has no custom modifiers.

Fields

PulseSale contract has following custom fields and constants:

- IERC20 public Pulse
- uint256 private percentageDivisor
- uint256 public totalStaked = 0
- uint256 public totalRewards = 0
- mapping (address => mapping(uint256 => UserStakeBracketInfo)) public stakes
- mapping (uint256 => uint256) public bracketDays
- mapping (uint256 => uint256) public stakeReward
- mapping (uint256 => uint256) public totalStakedInBracket
- mapping (uint256 => uint256) public totalRewardsInBracket
- mapping (address => bool) public Staked

Functions

PulseSale has following external or public functions:

- *constructor*
  Description
  Deploys the contract.
  Visibility
  public
  Input parameters
  - address _pulse

  Constraints
  None
  Events emit
  None
  Output
  None
- *stake14, stake1mo, stake3mo, stake6mo, stake12mo*
  Description
  Stakes an `_amount` for a corresponding time.
  Visibility
  public
  Input parameters
  - uint256 _amount

Constraints
- `nonReentrant` modifier
- A caller should not have any active stakes.
- An `_amount` should be at least 1 token.
- The Pulse balance of the contract should be enough to cover withdrawal of all existing stakes plus rewards.
- An allowance for at least `amount` should be set by a caller for the contract.

Events emit
Emits the `userStaked` event.
Output
None

- *claim14, claim1mo, claim3mo, claim6mo, claim12mo*
Description
Claims tokens.
Visibility
public
Input parameters
None
Constraints
- `nonReentrant` modifier
- A caller should have active stake.
- A payday of a stake should be reached.

Events emit
Emits the `userStaked` event.
Output
None

- *calculateReward, totalOwedValue, owedBalance*
Description
Simple view functions.

- *reclaimPulse*
Description
Allows owners to withdraw tokens that are not reserved.
Visibility
public
Input parameters
- uint256 _amount

Constraints
- `onlyOwner` modifier
- At least reserved amount of tokens should be remained on the contract.

Events emit
None
Output
None

- *changeReturnRateForBracket*

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

Description
Changes a reward for a `_stakeBracket`.
Visibility
public
Input parameters
- uint256 _percentage, uint256 _stakeBracket

Constraints
- `onlyOwner` modifier

Events emit
Emits the `stakeRewardUpdated` event.
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

1. The `_forwardFunds` function of the `PulseSale` contract transfers ETH using the `transfer` function. If the `_wallet` is a contract with fallback function, transfer will fail due to lack of gas.

   The Customer acknowledged with the possible issue and confirmed that the `_wallet` address will never be a contract.

Medium

1. Constructor of the `Pulse` contract has payable keyword but the contract is not designed to work with ETH. If any funds will be transferred during the contract deployment, those funds will be locked.

   We recommend removing payable keyword.

   Fixed before the second review.

2. The `Pulse` token inherits a copy of the OpenZeppelin ERC-20 contract.

   We recommend importing it directly from the OpenZepplein and not to store the local copy.

   In order to operate with the balances mapping, the local copy differs from the OpenZeppelin version. Visibility level of `_balances` storage increased to internal.

3. Imports of the `hardhat/console.sol` are redundant because it's never used. Also, its code is not provided and is out of the audit scope.

   Fixed before the second review.

4. `_postValidatePurchase` and `_updatePurchasingState` functions of the PulseSale contract are not implemented and can be removed.

   Fixed before the second review.

5. The `withdrawETH` function of the `PulseStake` is redundant as soon as the contract has no payable functions and may not accept ETH.

   Fixed before the second review.

Low

1. The `changeReturnRateForBracket` function of the `PulseStake` contract has no `_stakeBracket` value validation.

   Fixed before the second review.

Lowest / Code style / Best Practice

1. `feeReceiverChanged`, `userStaked`, `userClaimed`, `stakeRewardUpdated` events are not using CapWords naming style.
2. `Pulse` and `Staked` field of the `PulseStake` contract are not in a camelCase.

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 **1** high, **5** medium, **1** low and **2** informational issues during the audit.

After the second review, Customers` smart contracts contains **2** informational issues.

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.