# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer**: Just Liquidity
**Date**: November 14th, 2020

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 Just Liquidity (19 pages) |
| **Approved by** | Andrew Matiukhin \| CTO Hacken OU |
| **Type** | Limit order contracts. |
| **Platform** | Ethereum / Solidity |
| **Methods** | Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review |
| **Repository** | https://github.com/JustLiquidity/limit-orders/ |
| **Commit** | b80b37a0b60e3331976d4ea8bd072b1eb0e2f3a4 |
| **Deployed contract** | LimitOrderCore -<br>https://bscscan.com/address/0x22CCc580eB87C3B90126a71Fc2DF72449318451f<br>LimitOrders -<br>https://bscscan.com/address/0xa4410e6891245100f1dd4b57e2d631dbc1267cf3 |
| **Timeline** | 14TH NOV 2020 - 17TH NOV 2020 |
| **Changelog** | 17TH NOV 2020 - INITIAL AUDIT |

# Table of contents

# Introduction

Hacken OÜ (Consultant) was contracted by Just Liquidity (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 November 14th, 2020 - November 17th, 2020.

# Scope

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

```
Repository   https://github.com/JustLiquidity/limit-orders/
Commit b80b37a0b60e3331976d4ea8bd072b1eb0e2f3a4
Files:
        BSCswapHandler.sol
        LimitOrderCore.sol
        LimitOrders.sol
```

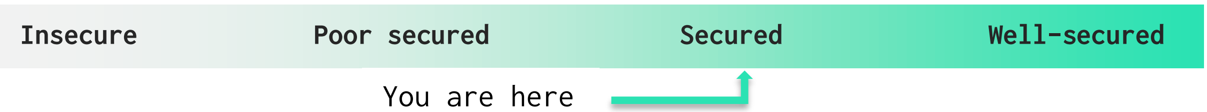
We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

| Category | Check Item |
|---|---|
| Code review | ▪ Reentrancy<br>▪ Ownership Takeover<br>▪ Timestamp Dependence<br>▪ Gas Limit and Loops<br>▪ DoS with (Unexpected) Throw<br>▪ DoS with Block Gas Limit<br>▪ Transaction-Ordering Dependence<br>▪ Style guide violation<br>▪ Costly Loop<br>▪ ERC20 API violation<br>▪ Unchecked external call<br>▪ Unchecked math<br>▪ Unsafe type inference<br>▪ Implicit visibility level<br>▪ Deployment Consistency<br>▪ Repository Consistency<br>▪ Data Consistency |

| | |
|---|---|
| Functional review | ▪ Business Logics Review<br>▪ Functionality Checks<br>▪ Access Control & Authorization<br>▪ Escrow manipulation<br>▪ Token Supply manipulation<br>▪ Assets integrity<br>▪ User Balances manipulation<br>▪ Data Consistency manipulation<br>▪ Kill-Switch Mechanism<br>▪ Operation Trails & Event Generation |

## Executive Summary

According to the assessment, the Customer's smart contracts are secure and can be used in production. Though, gas consumption can be optimized.



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 **2** medium and **3** low severity issues during the audit.

## 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. |

# AS-IS overview

## BSCswapHandler.sol

### Description

*BSCswapHandler* is a contract used to execute an order.

### Imports

*BSCswapHandler* contract has following imports from the repository:

- IWBNB
- IHandler
- IBSCswapPair
- UniswapUtils
- LimitOrderUtils
- SafeMath
- SafeERC20

### Inheritance

*BSCswapHandler* contract is IHandler.

### Usages

*BSCswapHandler* contract has following usages:

- *SafeMath for uint256*

### Structs

*BSCswapHandler* contract has no custom data structures.

### Enums

*BSCswapHandler* contract has no custom enums.

### Events

*BSCswapHandler* contract has no custom events

### Modifiers

*BSCswapHandler* has no custom modifiers.

### Fields

*BSCswapHandler* contract has following fields and constants:

- IWBNB public immutable WBNB;
- address public immutable FACTORY;

- bytes32 public immutable FACTORY_CODE_HASH;

**Functions**

*BSCswapHandler* has following public functions:

- ***constructor***
  **Description**
  Initializes the contract. Sets factory address, wbnb address and codeHash of the uniswap factory.
  **Visibility**
  public
  **Input parameters**
  - _factory - and address of the uniswap v2 factory contract.
  - _wbnb - Address of WBNB contract.
  - _codeHash - Bytes32 of the uniswap v2 pair contract unit code hash.

  **Constraints**
  None
  **Events emit**
  None
  **Output**
  None
- ***receive***
  **Description**
  Allows to receive eth from other contracts.
- ***handle***
  **Description**
  Handle an order execution.
  **Visibility**
  external payable
  **Input parameters**
  - _inputToken - Address of the input token
  - _outputToken - Address of the output token
  - uint256 - unused and unnamed argument
  - uint256 - unused and unnamed argument
  - _data - Bytes of arbitrary data

  **Constraints**
  None
  **Events emit**
  None
  **Output**
  - bought - Amount of output token bought
- ***canHandle***
  **Description**

Check whether it’s possible to handle an order execution.
**Visibility**
external view
**Input parameters**
  - _inputToken - Address of the input token
  - _outputToken - Address of the output token
  - _inputAmount - uint256 of the input token amount
  - _minReturn - uint256 of the min return amount of output token
  - _data - Bytes of arbitrary data

**Constraints**
None
**Events emit**
None
**Output**
  - bool - Whether the execution can be handled or not

- ***simulate***
  **Description**
  Simulate an order execution
  **Visibility**
  external view
  **Input parameters**
    - _inputToken - Address of the input token
    - _outputToken - Address of the output token
    - _inputAmount - uint256 of the input token amount
    - _minReturn - uint256 of the min return amount of output token
    - _data - Bytes of arbitrary data

  **Constraints**
  None
  **Events emit**
  None
  **Output**
    - bool - Whether the execution can be handled or not
    - uint256 - Amount of output token bought

## LimitOrders.sol

### Description

*LimitOrders* is a contract used to execute an order.

### Imports

*LimitOrders* contract has following imports from the repository:

- IModule
- IHandler
- Order

- SafeMath
- SafeERC20
- LimitOrderUtils

### Inheritance

*LimitOrders* contract is IModule and Order.

### Usages

*LimitOrders* contract has following usages:

- *SafeMath for uint256*

### Structs

*LimitOrders* contract has no custom data structures.

### Enums

*LimitOrders* contract has no custom enums.

### Events

*LimitOrders* contract has no custom events

### Modifiers

*LimitOrders* has no custom modifiers.

### Fields

*LimitOrders* contract has no custom fields and constants.

### Functions

*LimitOrders* has following public functions:

- ***receive***
  **Description**
  Allows to receive eth from other contracts.
- ***execute***
  **Description**
  Executes an order.
  **Visibility**
  external
  **Input parameters**
  - _inputToken - Address of the input token
  - _inputAmount - uint256 of the input token amount (order amount)
  - _data - Bytes of the order's data

- _auxData - Bytes of the auxiliar data used for the handlers to execute the order

**Constraints**
None
**Events emit**
None
**Output**
- bought - amount of output token bought

- *canExecute*
**Description**
Check whether an order can be executed or not.
**Visibility**
external view
**Input parameters**
  - _inputToken - Address of the input token
  - _inputAmount - uint256 of the input token amount (order amount)
  - _data - Bytes of the order's data
  - _auxData - Bytes of the auxiliar data used for the handlers to execute the order

**Constraints**
None
**Events emit**
None
**Output**
  - bool - whether the order can be executed or not

## LimitOrderCore.sol

### Description

*LimitOrderCore* is a contract used to create, execute and cancel orders. It’s only possible to use ETH (BNB) as an input token when placing an order.

### Imports

*LimitOrderCore* contract has following imports from the repository:

- SafeMath
- ECDSA
- Fabric
- IModule
- IERC20
- Order

### Inheritance

*LimitOrderCore* contract is Order.

### Usages

*LimitOrderCore* contract has following usages:

- *SafeMath for uint256*
- *using Fabric for bytes32;*

### Structs

*LimitOrderCore* contract has no custom data structures.

### Enums

*LimitOrderCore* contract has no custom enums.

### Events

*LimitOrderCore* contract has following events:

- *event* DepositETH(bytes32 indexed _key, address indexed _caller, uint256 _amount, bytes _data);
- event OrderExecuted(bytes32 indexed _key, address _inputToken, address _owner, address _witness, bytes _data, bytes _auxData, uint256 _amount, uint256 _bought);
- event OrderCancelled(bytes32 indexed _key, address _inputToken, address _owner, address _witness, bytes _data, uint256 _amount);

### Modifiers

*LimitOrderCore* has no custom modifiers.

### Fields

*LimitOrderCore* contract has following fields and constants:

- mapping(bytes32 => uint256) public ethDeposits;

### Functions

*LimitOrderCore* has following public functions:

- ***receive***
  **Description**
  Allows to receive eth from other contracts.
- ***depositEth***
  **Description**
  Create ETH (BNB) token order.
  **Visibility**

external
**Input parameters**
- _data - encoded order

**Constraints**
- An input token should be BNB

**Events emit**
Emits the `DepositETH` event.
**Output**
- bought - amount of output token bought

- *cancelOrder*
  **Description**
  Cancel an order
  **Visibility**
  external
  **Input parameters**
  - IModule _module - an address of a module to use for the order execution.
  - IERC20 _inputToken - address of the input token.
  - address payable _owner - address of the order's owner.
  - address _witness - address of the witness.
  - bytes calldata _data - encoded order data.

  **Constraints**
  - A msg.sender should be an order owner.

  **Events emit**
  Emits the `OrderCancelled` event.
  **Output**
  None
- *encodeEthOrder*
  **Description**
  Cancel an order
  **Visibility**
  external pure
  **Input parameters**
  - address _module - an address of a module to use for the order execution.
  - address _inputToken - address of the input token.
  - address payable _owner - address of the order's owner.
  - address _witness - address of the witness.
  - bytes calldata _data - encoded order data.
  - bytes32 _secret - private key of the _witness

  **Constraints**
  None
  **Events emit**
  None
  **Output**

  - bytes - input data to send the transaction

- *decodeOrder*
  **Description**
  Decode encoded order.
  **Visibility**
  public pure
  **Input parameters**
  - _data - Bytes of the order

  **Constraints**
  None
  **Events emit**
  None
  **Output**
  - bytes - input data to send the transaction
  - module - an address of a module to use for the order execution.
  - inputToken - address of the input token.
  - owner - address of the order's owner.
  - witness - address of the witness.
  - data - encoded order data.
  - secret - private key of the _witness
- *vaultOfOrder*
  **Description**
  Get a vault address of an order token.
  **Visibility**
  public view
  **Input parameters**
  - _module - Address of the module to use for the order execution
  - _inputToken - Address of the input token
  - _owner - Address of the order's owner
  - _witness - Address of the witness
  - _data - Bytes of the order's data

  **Constraints**
  None
  **Events emit**
  None
  **Output**
  - address - the vault address.
- *executeOrder*
  **Description**
  Executes an order.
  **Visibility**
  external
  **Input parameters**

- _module - Address of the module to use for the order execution
- _inputToken - Address of the input token
- _owner - Address of the order's owner
- _data - Bytes of the order's data
- _signature - Signature to calculate the witness
- _auxData - Bytes of the auxiliar data used for the handlers to execute the order

**Constraints**
None
**Events emit**
Emits the `OrderExecuted` event.
**Output**
None

- *existOrder*
  **Description**
  Check whether an order exists or not.
  **Visibility**
  external view
  **Input parameters**
  - _module - Address of the module to use for the order execution
  - _inputToken - Address of the input token
  - _owner - Address of the order's owner
  - _witness - Address of the witness
  - _data - Bytes of the order's data

  **Constraints**
  None
  **Events emit**
  None
  **Output**
  - bool - whether the order exists or not
- *canExecuteOrder*
  **Description**
  Check whether an order can be executed or not.
  **Visibility**
  external view
  **Input parameters**
  - _module - Address of the module to use for the order execution
  - _inputToken - Address of the input token
  - _owner - Address of the order's owner
  - _witness - Address of the witness
  - _data - Bytes of the order's data

- _auxData - Bytes of the auxiliar data used for the handlers to execute the order

**Constraints**

None

**Events emit**

None

**Output**

- bool - whether the order can be executed or not

- *keyOf*

  **Description**

  Get an order key.

  **Visibility**

  public pure

  **Input parameters**

  - _module - Address of the module to use for the order execution
  - _inputToken - Address of the input token
  - _owner - Address of the order's owner
  - _witness - Address of the witness
  - _data - Bytes of the order's data

  **Constraints**

  None

  **Events emit**

  None

  **Output**

  - bytes32 - order's key

## Audit overview

### 🟥🟥🟥🟥 Critical

No critical issues were found.

### 🟨🟨🟨 High

No high severity issues were found.

### 🟩🟩 Medium

1. Local versions of OpenZeppelin libraries are used. We recommend to import from the repositories to ensure their correctness and secureness.

2. The `LimitOrders` does not provide any complex logic and can be merged with the `LimitOrderCore` contract to decrease gas consumption.

### 🟦 Low

1. The `handle` function of the `BSCswapHandler` contract has redundant unused parameters.

2. The `execute` function of the `LimitOrders` contract has redundant unused parameters.

3. The Order contract contains only 1 constant and its functionality does not match its name. We recommend to remove this contract.

### 🟦 Lowest / Code style / Best Practice

No lowest severity issues were found.

# 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.

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 **2** medium and **3** low severity issues during the audit.

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

| Category | Check Item | Comments |
|---|---|---|
| Code review | ▪ Unused code | ▪ Unused variables can be found in the code. |
| | ▪ Gas Limit and Loops | ▪ Gas consumption is unjustifiably high for such kind of operations. |

# 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.