SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Unilayer
Date: October 30th, 2020

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 Unbound
Type	Token swap
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Contract Address	0xFA311750A0E1D2B8B979678EC1A04F56AC8DB866
Creator Txn Hash	0xBE5C948D39D06FF11612B97FAFBFC5804D8ABDEAA00D1A7EA9272D8B7D898F6C
Timeline	26TH OCT 2020 - 30TH OCT 2020
Changelog	28TH OCT 2020 - Initial Audit 30TH OCT 2020 - Fixes Check
Approved by	Andrew Matiukhin | CTO Hacken OU

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

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Unilayer (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 26th, 2020 - October 28th, 2020.

Scope

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

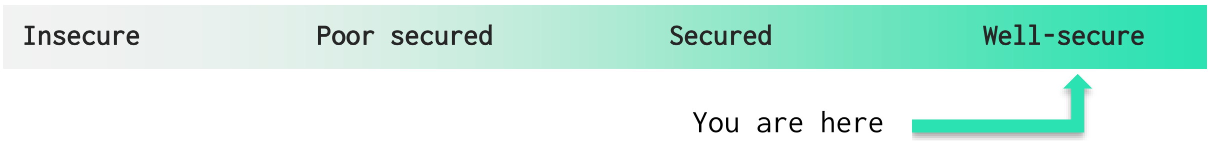
```
Contract Address – 0xFa311750A0E1d2b8B979678Ec1A04F56aC8DB866
Creator Txn Hash -
0xbe5c948d39d06ff11612b97fafbfc5804d8abdeaa00d1a7ea9272d8b7d898f6c
```

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

Executive Summary

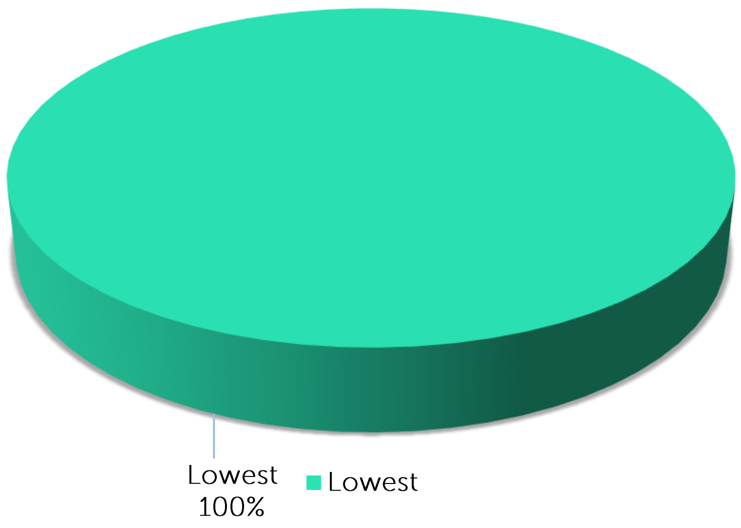
According to the assessment, the Customer's smart contracts do not have high vulnerability and can be considered secure. Some fixes are recommended though.



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 lowest severity issue during audit.

Graph 1. The distribution of vulnerabilities.



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.

AS-IS overview

unilayer.sol

Description

UniLayer is a contract for operating swap orders.

Imports

UniLayer contract has 7 imports:

- *SafeMath* – from OpenZeppelin;
- *TransferHelper* – from Uniswap;
- *IUniswapV2Factory* – from Uniswap;
- *IUniswapV2Router01* – from Uniswap;
- *IUniswapV2Router02* – from Uniswap;
- *Context* – from OpenZeppelin;
- *Ownable* – from OpenZeppelin;

Inheritance

UniLayer contract inherits *Ownable*.

Usings

UniLayer contract use:

- *SafeMath* for *uint256*;

Enums

UniLayer contract has 2 enums:

- *enum OrderState {Created, Cancelled, Finished}*
- *enum OrderType {EthForTokens, TokensForEth, TokensForTokens}*

Structs

UniLayer contract has 1 data struct:

- *Order*
    - *OrderState orderState* – an order state;
    - *OrderType orderType* – an order type;
    - *address payable traderAddress* – an address of trader;

- *address assetIn* – an address of asset in;
- *address assetOut* – an address of asset out;
- *uint assetInOffered* – an amount of asset in;
- *uint assetOutExpected* – an amount of asset out;
- *uint executorFee* – an executor fee;
- *uint stake* – a stake value;
- *uint id* – the id of an order;
- *uint ordersI* – the index of an order;

Fields

UniLayer contract has 9 fields:

- *IUniswapV2Router02 public immutable uniswapV2Router* – an interface of Uniswap Router V2;
- *IUniswapV2Factory public immutable uniswapV2Factory* – an interface of Uniswap Factory V2;
- *uint public STAKE_FEE* – a multiplier of stake fee;
- *uint public EXECUTOR_FEE* – an executor fee;
- *uint[] public orders* – a list of active orders ids;
- *uint public ordersNum* – the next order id;
- *address public stakeAddress* – an address of stake;
- *mapping(uint => Order) public orderBook* – an order book;
- *mapping(address => uint[]) private ordersForAddress* – a mapping of orders ids for an address;

Functions

UniLayer has 14 functions:

- ***constructor***

  Description

  Initializes contract.
  Sets *uniswapV2Router*, *uniswapV2Factory* fields.

  Visibility

  public

  Input parameters

  - *IUniswapV2Router02 _uniswapV2Router* – an interface for Uniswap router V2;

  Constraints

None

Events emit

None

Output

None

- *setNewStakeFee*

Description

Sets stake fee.

Visibility

external

Input parameters

  - *uint256 _STAKE_FEE* – a stake fee;

Constraints

  - Only Owner can call it.
  - Fee value mast be greater or equal 0.

Events emit

None

Output

None

- *setNewExecutorFee*

Description

Sets executor fee.

Visibility

external

Input parameters

- *uint256 _EXECUTOR_FEE* – an executor fee;

Constraints

- Only Owner can call it.
- Fee value mast be greater or equal 0.

Events emit

None

Output

None

- *setNewStakeAddress*

Description

Sets stake address.

Visibility

external

Input parameters

- *address _stakeAddress* – an address of stake;

Constraints

- Only Owner can call it.
- Stake address can not be 0.

Events emit

None

Output

None

- *getPair*

Description

Gets Uniswap pair.

Visibility

internal view

Input parameters

- *address tokenA* – an address of token;
- *address tokenB* – an address of token;

Constraints

- Token pair is available.

Events emit

None

Output

Returns pair address.

- *updateOrder*

Description

Updates an order.

Visibility

internal

Input parameters

- *Order memory order* – an order;
- *OrderState newState* – a new state of order;

Constraints

None

Events emit

None

Output

None

- *createOrder*

  Description

  Creates an order.

  Visibility

  external payable

  Input parameters

  - *OrderType orderType* – a type of order;
  - *address assetIn* – an address of asset in;
  - *address assetOut* – an address of asset out
  - *uint assetInOffered* – an amount of asset in;
  - *uint assetOutExpected* – an amount of asset out;
  - *uint executorFee* – an executor fee;

  Constraints

  - Asset in amount must be greater than 0.
  - Asset out amount must be greater than 0.
  - *executorFee* must be greater or equal *EXECUTOR_FEE*.
  - WETH as the *assetIn* must used for *EthForTokens* order type.
  - Transaction value must cover asset in amount and all fees for *EthForTokens* order type.
  - Transaction value must match *executorFee*.
  - WETH as the *assetOut* must used for *TokensForEth* order type.

  Events emit

  - *logOrderCreated*

  Output

  None

- *executeOrder*

  Description

  Executes the order.

  Visibility

external

Input parameters

- *uint orderId* the id of the order;

Constraints

- *traderAddress* can not be 0.
- Order state must be *Created*.

Events emit

- *logOrderExecuted*

Output

Returns swap result.

- ***cancelOrder***

Description

Cancels the order.

Visibility

external

Input parameters

- *uint orderId* the id of the order;

Constraints

- *traderAddress* can not be 0.
- Only *traderAddress* can call it.
- Order state must be *Created*.

Events emit

- *logOrderCancelled*

Output

None

- ***calculatePaymentETH***

Description

Calculates ETH payments.

Visibility

external view

Input parameters

- *uint ethValue* – an amount of ETH;

Constraints

None

Events emit

None

Output

Returns ETH payments.

- *getOrdersLength*

Description

Gets orders length.

Visibility

external view

Input parameters

None

Constraints

None

Events emit

None

Output

Returns orders length.

- *getOrdersId*

  Description

  Gets the id of the order.

  Visibility

  external view

  Input parameters

  - *uint i* – an index of order;

  Constraints

  None

  Events emit

  None

  Output

  Returns the id.

- *getOrdersForAddressLength*

  Description

  Gets the length of orders for the address.

  Visibility

  external view

  Input parameters

  - *address _address* – an address;

  Constraints

  None

  Events emit

None

Output

Returns the length of orders for the address.

- *getOrderIdForAddress*

Description

Gets the id of the order for the address.

Visibility

external view

Input parameters

  - *address _address* – an address;
  - *uint index* – an index;

Constraints

None

Events emit

None

Output

Returns the id.

Audit overview

🟥🟥🟥🟥 Critical

No critical issues were found.

🟨🟨🟨 High

No high severity issues were found.

🟩🟩 Medium

No medium severity issues were found.

🟦 Low

No medium severity issues were found.

🟦 Lowest / Code style / Best Practice

1. *setNewStakePercentage* has unnecessary check. The minimum value for uint256 is 0.

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.

Violations in following categories were found and addressed to Customer:

Category	Check Item	Comments
Code review	▪ Functionality Checks	*setNewStakePercentage* don't need to check that value bigger than 0.

Security engineers found 1 lowest severity issues during audit. It is recommended to fix it.

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.