SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: TrustSwap
Date: March 7th, 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 fixed - upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for TrustSwap - Draft
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Vesting
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/trustswap/trustswap-vest-contracts
Commit	d5af6ec8a723c515d604f2dd329849f0ed9917ea
Deployed contract	
Timeline	04 March 2021 - 07 March 2021
Changelog	07 FEB 2021 - INITIAL AUDIT

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by TrustSwap (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 March 7th, 2021.

Scope

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

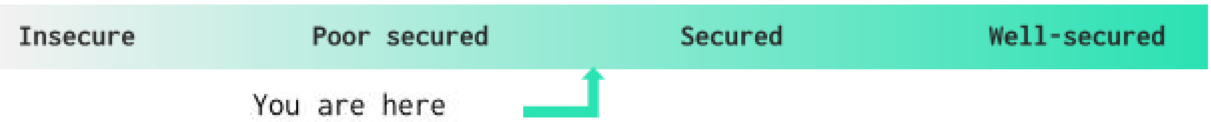
Contract deployment address:
Repository: https://github.com/trustswap/trustswap-vest-contracts
d5af6ec8a723c515d604f2dd329849f0ed9917ea
SwapTokenLocker.sol
SwapAdmin.sol
SwapTokenLockerFactory.sol

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
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

Executive Summary

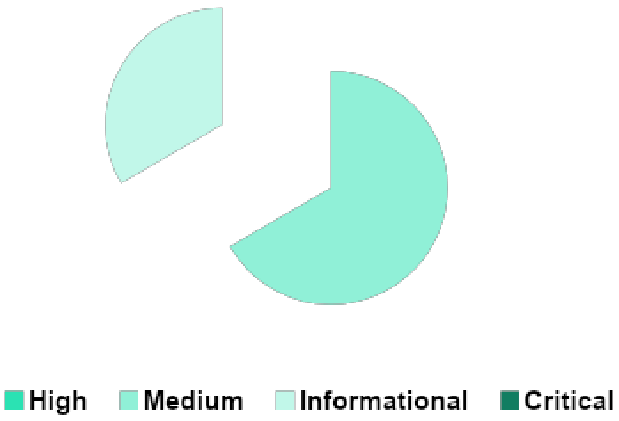
According to the assessment, the Customer's smart is secure.



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, **1** informational issue during first review.

Graph 1. The distribution of vulnerabilities after the first review.



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

SwapAdmin.sol

Description

SwapAdmin is a contract used to control admin access.

Imports

SwapAdmin contract no imports.

Inheritance

SwapAdmin contract does not inherit anything.

Usages

SwapAdmin contract has no usages.

Structs

SwapAdmin contract has no custom data structures.

Enums

SwapAdmin contract has no custom enums.

Events

SwapAdmin contract has following events:

- candidateChanged
- AdminChanged

Modifiers

SwapAdmin has following modifiers:

- onlyAdmin - checks whether a msg.sender is admin.

Fields

SwapAdmin contract has following custom fields and constants:

- address public admin
- address public candidate

Functions

SwapAdmin has following external or public functions:

- *constructor*
  Description
  Deploys the contract. Sets current admin.
  Visibility
  public
  Input parameters
    - address _admin
  Constraints
  None
  Events emit
  Emits the *AdminChanged* event.
  Output
  None
- *setCandidate*
  Description
  Sets a new admin candidate.
  Visibility
  external
  Input parameters
    - address _candidate
  Constraints
    - onlyAdmin modifier
  Events emit
  Emits the *candidateChanged* event.
  Output
  None
- *becomeAdmin*
  Description
  Accepts admin permissions.
  Visibility
  external
  Input parameters
  None
  Constraints
    - A message sender should be *candidate.*
  Events emit
  Emits the *AdminChanged* event.
  Output
  None

SwapTokenLocker.sol

Description

SwapTokenLocker is a tokens lock.

Imports

SwapTokenLocker contract following imports:

- @openzeppelin/contracts-ethereum-package/contracts/token/ERC20/IERC20.sol
- @openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol
- @openzeppelin/contracts/utils/Pausable.sol
- ./SwapAdmin.sol

Inheritance

SwapTokenLocker contract is SwapAdmin, Pausable.

Usages

SwapTokenLocker contract has following usages:

- SafeMath for uint

Structs

SwapTokenLocker contract has following data structures:

- LockInfo - stores lock info.

Enums

SwapTokenLocker contract has no custom enums.

Events

SwapTokenLocker contract has no events.

Modifiers

SwapTokenLocker has no custom modifiers.

Fields

SwapTokenLocker contract has following custom fields and constants:

- mapping (address => mapping(address => LockInfo)) public lockData
- mapping (address => address[]) public claimableTokens

Functions

SwapTokenLocker has following external or public functions:

- ***constructor***
  Description
  Deploys the contract. Sets current admin.
  Visibility
  public
  Input parameters

  - address _admin

Constraints

None

Events emit

Emits the *AdminChanged* event.

Output

None

- ***emergencyWithdraw***

  Description

  Allows admin to withdraw tokens.

  Visibility

  external

  Input parameters

  - address _tokenAddress

  Constraints

  - *onlyAdmin* modifier
  - *_tokenAddress* should not be zero

  Events emit

  None

  Output

  None

- ***getLockData, getClaimableTokens***

  Description

  Simple view functions.

- ***sendLockTokenMany***

  Description

  Locks multiple tokens.

  Visibility

  external

  Input parameters

  - address[] calldata _users
  - address[] calldata _tokenAddresses
  - uint256[] calldata _amounts
  - uint256[] calldata _lockTimestamps
  - uint256[] calldata _lockHours

  Constraints

  - *onlyAdmin* modifier
  - Length of all input arrays should be equal

  Events emit

  None

  Output

  None

- ***sendLockToken***

  Description

  Locks an *_amount* of tokens for a *_user.*

  Visibility

  external

  Input parameters

  - address _user

- address _tokenAddress
- uint256 _amount
- uint256 _lockTimestamp
- uint256 _lockHours

Constraints

- *onlyAdmin* modifier
- All input parameters should not be 0 address.

Events emit

None

Output

bool

- *claimToken*

Description

Claims an *_amount* of tokens.

Visibility

external

Input parameters

- uint256 _amount
- address _tokenAddress

Constraints

- All input parameters should not be 0 address.
- The lock timestamp should exceed a current block timestamp.
- The lock amount should be greater than 0.
- Hours passed after the lock start should be greater than 0.
- Available tokens amount should be greater than 0.

Events emit

None

Output

Unit256 - claimed amount.

Audit overview

🟥 🟥 🟥 🟥 Critical

No Critical severity issues were found.

🟨 🟨 🟨 High

No High severity issues were found.

🟩 🟩 Medium

1. **File**: SwapTokenLockerFactory

Method: getLastDeployed(address)

An assertion violation was triggered.
The underflow issue is possible when the owner didn't deploy any contract

In file: contracts/SwapTokenLockerFactory.sol:12

```
    uint256 length =
deployedContracts[owner].length;
    deployedContracts[owner][length - 1]
```

2. **File**: SwapTokenLocker

Method: sendLockTokenMany(address[] calldata _users, uint128[] calldata _amounts, uint32[] calldata _lockHours, uint256 _sendAmount)

A reentrancy attack can occur when you create a function that makes an external call to another untrusted contract before it resolves any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the effects were resolved. Please consider to use `zeppelin-solidity/ReentrancyGuard.sol` to protect the code from reentrancy.

In file: contracts/SwapTokenLocker.sol:48

```
    IERC20(token).transferFrom(msg.sender, address(this),
_sendAmount);
    for (uint256 j = 0; j < _users.length; j++) {
        sendLockToken(_users[j], _amounts[j],
uint64(block.timestamp), _lockHours[j]);
    }
```

🟦 Low

No low severity issues were found.

■ Lowest / Code style / Best Practice

A bunch of redundant overflow checks can be removed

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 **2** medium, **1** informational issue during the audit.

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.