# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer**: TosDis Finance
**Date**: Jan 14th, 2021

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 Tosdis Finance |
| **Approved by** | Andrew Matiukhin \| CTO Hacken OU |
| **Type** | Token, Token sale, Exchange, Exchanges aggregator. |
| **Platform** | Ethereum / Solidity |
| **Methods** | Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review. |
| **Repository** | |
| **Commit** | |
| **Deployed contract** | |
| **Timeline** | Jan 13th, 2021 – Jan 17th, 2021 |
| **Changelog** | Jan 14th,2021 - Initial Audit<br>Jan 17th,2021 - Remediation check |

# Table of contents

## Introduction

Hacken OÜ (Consultant) was contracted by (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 Jan 13th, 2021 - January 14th, 2021.

Remediation check was conducted Jan 17th, 2021

## Scope

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

```
Contract deployment address:
Repository
Commit
Files:
      DISToken.sol
      TokenVesting.sol
```

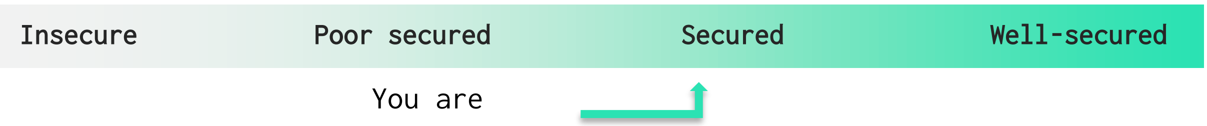
We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

| 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>▪ Kill-Switch Mechanism<br>▪ Operation Trails & Event Generation |

## Executive Summary

According to the assessment, the Customers' smart contracts are secure.



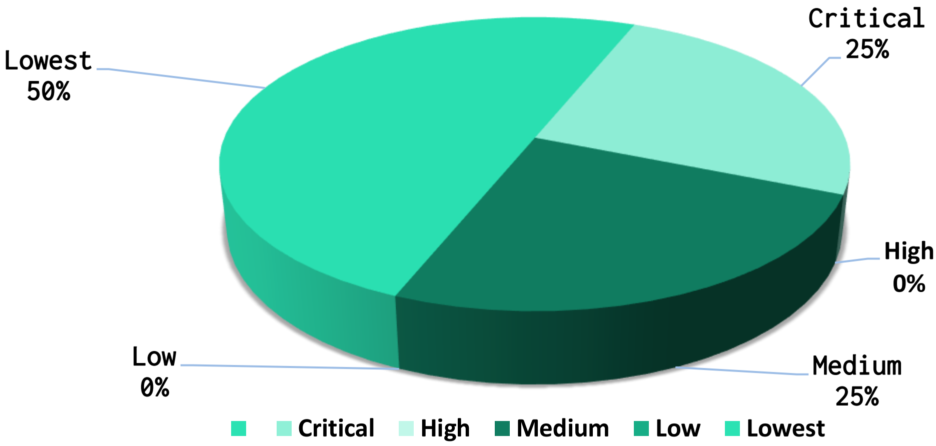
Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section, and all found issues can be found in the Audit overview section.

Security engineers found **1** critical, **1** medium and **2** informational issues during the audit.

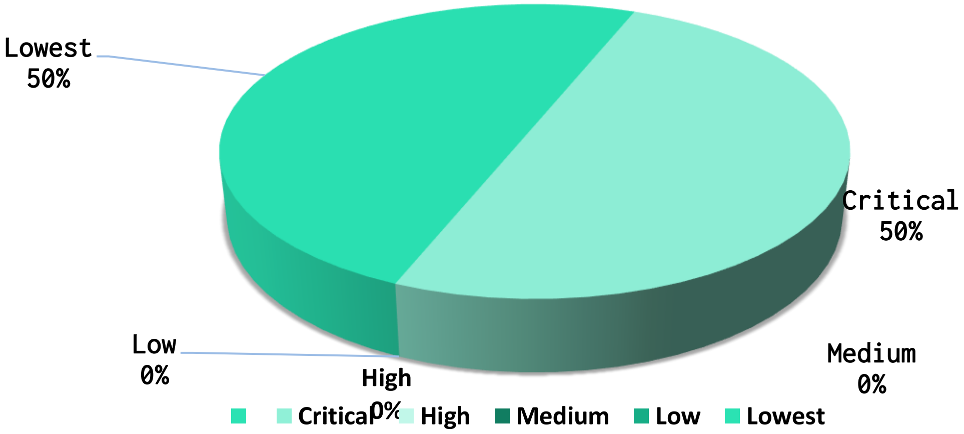
After the **second** review, Customers' smart contracts contain **1** critical vulnerability.

After the **third** review, Customers' smart contracts contains no issues.

*Graph 1. The distribution of vulnerabilities at initial audit*



*Graph 2. The distribution of vulnerabilities at 1st remediation check*



## 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

### DISToken.sol

#### Description

*DISToken* is a submitted code implements burnable ERC20 token. Token interfaces and implementations are inherited from OpenZeppelin Contracts. Token has unlimited supply and could be minted only by account with minter role.

#### Imports

*DISToken* contract has the following imports:

- import "@openzeppelin/contracts/access/AccessControl.sol";
- import "@openzeppelin/contracts/GSN/Context.sol";
- import "@openzeppelin/contracts/token/ERC20/ERC20Burnable.sol";

#### Usages

*DISToken* contract hasn’t the following custom usages.

#### Structs

*DISToken* contract has no the following data structures.

#### Enums

*DISToken* contract has no custom enums.

#### Events

*DISToken* contract hasn’t the following events.

#### Modifiers

*DISToken* has no custom modifiers.

#### Fields

*DISToken* contract hasn’t following constants.

#### Functions

*DISToken* has following public functions:

- *constructor*
  **Visibility**

public ERC20

**Input parameters**

- uint256 totalSupply,
- string memory name,
- string memory symbol,
- name,
- symbol.

**Constraints**

None

**Events emit**

None

**Output**

None

- *mint*

**Description**

**Visibility**

public virtual

**Input parameters**

- address to,
- uint256 amount.

**Constraints**

None

**Events emit**

None

**Output**

None

## TokenVesting.sol

### Description

*TokenVesting* is represents vault with vesting scheme inside. Each account has vesting with start time, duration and interval parameters. These are setting up for beneficiary during vesting creation. Beneficiary has ability to postpone start date, this function secured by requirements for start date and balance. Any account could invoke release function and withdraw appropriate amount of tokens. This amount calculated by releasableAmount view function.

### Imports

*TokenVesting* contract has the following imports:

- import "@openzeppelin/contracts/GSN/Context.sol";

- import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
- import "@openzeppelin/contracts/math/SafeMath.sol";

### Usages

*TokenVesting* contract has custom usages:

- SafeMath for uint256;

### Structs

*TokenVesting* contract has custom data structures:

- Vesting

### Enums

*TokenVesting* contract has no custom enums.

### Events

*TokenVesting* contract has the following events:

- Released(uint256 amount);

### Modifiers

*TokenVesting* has no the following modifiers.

### Fields

*TokenVesting* contract has following constants:

- IERC20 private _token;
- mapping (address => Vesting) private _vestings;

### Functions

*TokenVesting* has following public functions:

- *constructor*
  **Visibility**
  public
  **Input parameters**
    - address token
  **Constraints**
  None
  **Events emit**
  None

**Output**
None

- *getVesting*
  **Description**

  **Visibility**
  public view
  **Input parameters**
  - address beneficiary

  **Constraints**
  None
  **Events emit**
  None
  **Output**
  - uint256
  - uint256
  - uint256
  - uint256
  - uint256
- *createVesting*
  **Description**

  **Visibility**
  public
  **Input parameters**
  - address sender,
  - address beneficiary,
  - uint256 start,
  - uint256 interval,
  - uint256 duration,
  - uint256 amount

  **Constraints**
  None
  **Events emit**
  None
  **Output**
  None
- *postponeVesting*
  **Description**

  **Visibility**
  external

**Input parameters**
  - uint256 start

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

- *release*
  **Description**

  **Visibility**
  public
  **Input parameters**
    - address beneficiary

  **Constraints**
  None
  **Events emit**
    - Released(unreleased);

  **Output**
  None

- *releasableAmount*
  **Description**

  **Visibility**
  public view
  **Input parameters**
    - address beneficiary

  **Constraints**
  None
  **Events emit**
  None
  **Output**
    - uint256

- *vestedAmount*
  **Description**

  **Visibility**
  public view
  **Input parameters**
    - address beneficiary

  **Constraints**
  None.
  **Events emit**

None

**Output**

- uint256

## Audit overview

### 🟥🟥🟥🟥 Critical

1. Functions transfer and transferFrom are not checked for success and can return false value. Use SafeTransfer and SafeTransferFrom functions instead.

   ***Fixed before the second review.***

2. \`createVesting\` function allows specifying an account from where funds will be transferred. Such flow allows stealing funds from accounts that sent an allowance transaction but did not call the \`createVesting\` function yet.

   We recommend removing the \`sender\` parameter and use the message sender instead.

   ***Fixed before the third review.***

### 🟩🟩 Medium

1. Protect contacts by preventing of the reentrancy attack by Reentrancy Guard. Apply it to all public and external functions. Reentrancy Guard module helps to prevent reentrant calls to a function. Inheriting from ReentrancyGuard will make the nonReentrant modifier available and it could be applied to functions to make sure there are no nested (reentrant) calls to them.

   ***Fixed before the second review.***

### 🟦 Lowest / Code style / Best Practice

1. createVesting and release functions could be implemented as external instead of public.

   ***Fixed before the second review.***

## 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** critical, **1** medium and **2** informational issues during the audit.

After the **second** review, Customers' smart contracts contain 1 critical vulnerability.

After the **third** review, Customers' smart contracts contains no issues.

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

| Category | Check Item | Comments |
|---|---|---|
| Code review | ▪ Data Consistency | ▪ Data consistency can be violated. |
| | ▪ Business Logics Review | ▪ The source code was received without whitepaper. |
| | ▪ Style guide violation | ▪ Several minor code-style issues were found. |
| | ▪ Assets integrity | ▪ The transfer method can lock up all fund irreversibly. |
| | ▪ Reentrancy | ▪ Lack of reentrancy guard checks. |

# 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.