# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer**: Tosdis
**Date**: December 16th, 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 Tosdis Finance (22 pages). |
| **Approved by** | Andrew Matiukhin \| CTO Hacken OU |
| **Type** | Staking protocol |
| **Platform** | Ethereum / Solidity |
| **Methods** | Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review. |
| **Repository** | |
| **Commit** | |
| **Deployed contract** | |
| **Timeline** | Dec, 10th 2020 - Dec, 16th 2020 |
| **Changelog** | Dec, 10th 2020 - Dec, 12th 2020 Initial audit<br>Dec, 14th 2020 - Remediation check<br>Dec, 15th 2020 - Remediation check<br>Dec, 16th 2020 - Remediation check |

# Table of contents

## Introduction

Hacken OÜ (Consultant) was contracted by Tosdis (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:

- Smart contract audit conducted between December 10th, 2020 - December 16th, 2020.
- Remediation check was done December 14th, 2020;
- 2nd Remediation check was done December 15th, 2020;
- 3nd Remediation check was done December 16th, 2020.

## Scope

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

```
Contract deployment address:
Repository
Commit
Files:
      ERC20Basic.sol
      Migrations.sol
      Ownable.sol
      StakingPool.sol
      StakeMaster.sol
```

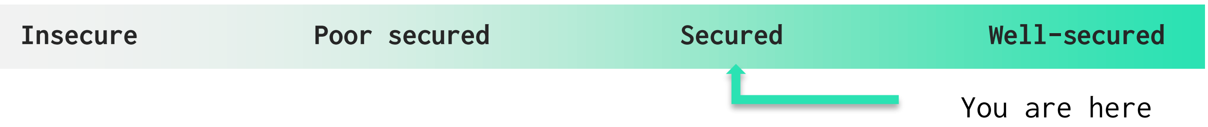
We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

| 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 Customer's smart has issues that should be fixed. The code quality should be increased.

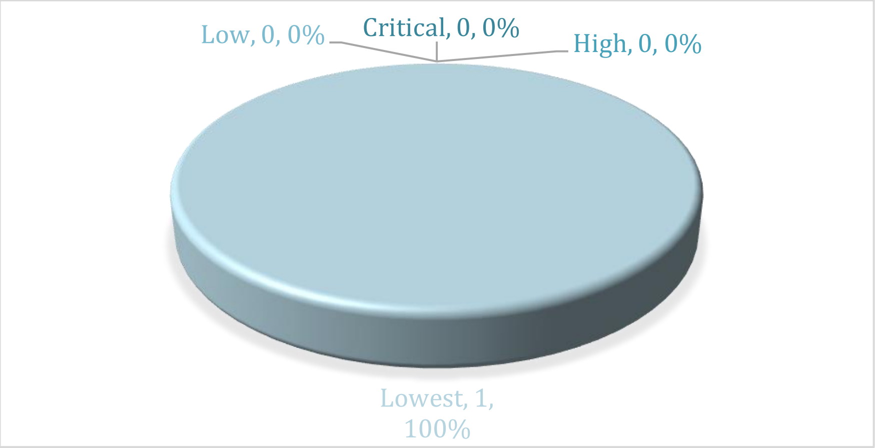


Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section,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

Security engineers found **1** critical **0** high, **0** medium, **0** low and **5** lowest issues during the first audit.

Update: Most contract’s vulnerabilities were fixed after the audit was done. One low lowest severities left in contract and this risk is acceptable. For details check “Audit overview” section.

***Graph 1. The distribution of vulnerabilities after 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

## ERC20Basic.sol

### Description

Basic contract ERC20. The basic functions are defined: totalSupply, transfer, approve, allowance and transferFrom. Used as an ERC20 token in StakingPool and StakeMaster contracts. Used in scripts for deploying StakingPool and StakeMaster contracts. Used in tests.

### Imports

*ERC20Basic* contract hasn't the imports.

### Usages

*ERC20Basic* contract has the following custom usages:

- SafeMath for uint256

### Variables:

- string public constant name = "ERC20Basic";
- string public constant symbol = "BSC";
- uint8 public constant decimals = 18;
- mapping(address => uint256) balances;
- mapping(address => mapping (address => uint256)) allowed;
- uint256 totalSupply_;

### Structs

*ERC20Basic* contract has the following data structures:

- It is not possible to define name, symbol and decimals in the constructor.

### Enums

*ERC20Basic* contract has no custom enums.

### Events

*ERC20Basic* contract has the following events:

- event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
- event Transfer(address indexed from, address indexed to, uint tokens);

**Modifiers**

ERC20Basic has no custom modifiers.

**Fields**

*ERC20Basic* contract has following constants:

- string public constant name = "ERC20Basic";
- string public constant symbol = "BSC";
- uint8 public constant decimals = 18;

**Functions**

*ERC20Basic* has following public functions:

- ***constructor***
  **Visibility**
  public
  **Input parameters**
  - uint256 total

  **Constraints**
  None
  **Events emit**
  None
  **Output**
  None
- ***totalSupply***
  **Visibility**
  Public view
  **Input parameters**
  None
  **Constraints**
  None.
  **Events emit**
  None
  **Output**
  Uint256
- ***transfer***
  **Visibility**

public
**Input parameters**
  - address receiver,
  - uint numTokens

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

- ***approve***
  **Visibility**
  public
  **Input parameters**
    - address delegate
    - uint numTokens

  **Constraints**
  None
  **Events emit**
  None
  **Output**
  Bool
- ***allowance***
  **Visibility**
  Public view
  **Input parameters**
    - address owner
    - address delegate

  **Constraints**
  None
  **Events emit**
  None
  **Output**
  uint
- ***transferFrom***
  **Visibility**
  public
  **Input parameters**
    - address owner,
    - address buyer,
    - uint numTokens

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

**Output**
Bool

## Migrations.sol

### Description

Used in deployment scripts.

### Variables

- address public owner = msg.sender;
- uint public last_completed_migration;

### Functions

*Ownable* has following public functions:

- setCompleted(uint completed) public restricted.

## Ownable.sol

### Description

*Ownable t*he contract has an owner's address and provides basic authorization control, making it easier to implement user permissions.

### Inheritance

*Ownable* contract is StakingPool, StakeMaster.

### Variables

- address private _owner

### Events

*Ownable* contract has the following custom events:

- event OwnershipTransferred(address indexed previousOwner, address indexed newOwner)

### Functions

*Ownable* has following public functions:

- constructor () internal;
- owner() public view returns (address);

- isOwner() public view returns (bool);
- renounceOwnership() public onlyOwner;
- transferOwnership(address newOwner) public onlyOwner;
- _transferOwnership(address newOwner) internal.

## StakingPool.sol

### Description

*StakingPoo* is contract for staking tokens stakingToken.

### Imports

*StakingPool* contract hasn't the imports.

### Usages

*StakingPool* contract has the following custom usages:

- SafeMath for uint;

### Variables

- IERC20 public stakingToken;
- IERC20 public rewardToken;
- uint256 public startBlock;
- uint256 public lastRewardBlock;
- uint256 public finishBlock;
- uint256 public totalShares;
- uint256 public rewardPerBlock;
- uint256 public accTokensPerShare; // Accumulated tokens per share
- mapping (address => uint256) public stakes;
- mapping (address => uint256) public rewardDebts.

### Structs

*StakingPool* contract has no custom data structures.

### Enums

*StakingPool* contract has no custom enums.

### Events

*StakingPool* contract has the following custom events:

- event FinishBlockUpdated(uint256 _newFinishBlock);
- event PoolReplenished(uint256 amount);
- event TokensStaked(address stakeholder, uint256 amount, uint256 sharesAchived);
- event StakeWithdrawn(address stakeholder, uint256 amount, uint256 reward);
- event EmergencyWithdraw(address indexed user, uint256 amount).

### Modifiers

*StakingPool* has the no custom modifiers.

### Fields

*StakingPool* contract hasn’t constants.

### Functions

*StakingPool* has following public functions:

- ***constructor***
  **Description**
  Defines stakingToken, rewardToken, startBlock, finishBlock, poolTokenAmount, rewardPerBlock. stakingToken and poolToken - IERC20 tokens.
  **Visibility**
  public
  **Input parameters**
    - address _stakingToken,
    - address _poolToken,
    - uint256 _startBlock,
    - uint256 _finishBlock,
    - uint256 _poolTokenAmount

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

- ***getMultiplier***
  **Description**
  Getting the current multiplier for calculating the reward.
  **Visibility**
  public view
  **Input parameters**
  - uint256 _from,
  - uint256 _to

  **Constraints**
  - depends on finishBlock number and from and to block numbers.

  **Events emit**
  None
  **Output**
  - uint256
- ***pendingReward***
  **Description**
  Calculates the current possible reward for the holder.
  **Visibility**
  external view
  **Input parameters**
  - address _user

  **Constraints**
  None
  **Events emit**
  None
  **Output**
  - uint256
- ***updatePool***
  **Description**
  Updates accTokensPerShare and lastRewardBlock accumulating rewards.
  **Visibility**
  public
  **Input parameters**
  None
  **Constraints**
  None
  **Events emit**
  None
  **Output**
  - accTokensPerShare
  - lastRewardBlock
- ***stakeTokens***
  **Description**

The user can stake a certain amount of coins, if he already has staked coins, the reward is calculated and rewardTokens are transferred to him.

**Visibility**

public

**Input parameters**

- uint256 _amountToStake

**Constraints**

- If amountToStake is greater than 0, stakingToken is deducted from the user .

**Events emit**

None

**Output**

- rewardDebts

- ***withdrawStake***

**Description**

The withdrawal amount is checked, the reward is calculated and sent to the user, withdrawn from the coin staking.

**Visibility**

public

**Input parameters**

- uint256 _stakeAmount

**Constraints**

None

**Events emit**

None

**Output**

None

- ***emergencyWithdraw***

**Description**

Line withdrawal, stakingToken, but no reward, it is replaced.

**Visibility**

public

**Input parameters**

None

**Constraints**

None

**Events emit**

None

**Output**

None

- ***emergencyRewardWithdraw***

**Description**

Urgent withdrawal of rewardToken by the owner.

**Visibility**

public
**Input parameters**
  - uint256 _amount

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

- ***setFinishBlock***
  **Description**
  Setting a higher staking end number by the owner.
  **Visibility**
  external onlyOwner
  **Input parameters**
    - uint256 _newFinishBlock

  **Constraints**
  None
  **Events emit**
  None
  **Output**
  uint256 public rewardPerBlock
- ***topUpStakingPool***
  **Description**
  Replenishment by rewardToken owner. The rewardPerBlock is recalculated.
  **Visibility**
  external onlyOwner
  **Input parameters**
    - uint256 _topUpAmount

  **Constraints**
  None
  **Events emit**
  None
  **Output**
  uint256 public rewardPerBlock

## StakeMaster.sol

### Description

*StakeMaster* is used to create the StakingPool.

### Imports

*StakeMaster* contract hasn’t the imports.

### Usages

*StakeMaster* contract has the following custom usages:

- SafeMath for uint;

### Variables

- IERC20 public feeToken;
- address public feeWallet;
- uint256 public feeAmount;
- uint256 public burnPercent;
- uint256 public divider.

### Structs

*StakeMaster* contract has no custom data structures.

### Enums

*StakeMaster* contract has no custom enums.

### Events

- event StakingPoolCreated(address owner, address pool);
- event TokenFeeUpdated(address newFeeToken);
- event FeeAmountUpdated(uint256 newFeeAmount);
- event BurnPercentUpdated(uint256 newBurnPercent);
- event FeeWalletUpdated(address newFeeWallet).

### Modifiers

*StakeMaster* has the no custom modifiers.

### Fields

*StakeMaster* contract hasn't constants.

### Functions

*StakeMaster* has following public functions:

- ***constructor***
  **Description**
  Defines the values of feeToken, feeWallet, feeAmount, burnPercent. feeToken - IERC20 token.
  **Visibility**

public

**Input parameters**

- address _feeToken,
- address _feeWallet,
- uint256 _feeAmount,
- uint256 _burnPercent.

**Constraints**

None

**Events emit**

None

**Output**

None

- ***setFeeToken***

  **Description**

  Defines a new feeToken.

  **Visibility**

  external onlyOwner

  **Input parameters**

  - address _newFeeToken

  **Constraints**

  None

  **Events emit**

  None

  **Output**

  None

- ***setFeeAmount***

  **Description**

  Defines a new feeAmount.

  **Visibility**

  external onlyOwner

  **Input parameters**

  - uint256 _newFeeAmount

  **Constraints**

  None

  **Events emit**

  None

  **Output**

  None

- ***setFeeWallet***

  **Description**

  Defines a new feeWallet.

  **Visibility**

  external onlyOwner

  **Input parameters**

  - address _newFeeWallet

  **Constraints**

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

- ***setBurnPercent***
  **Description**
  Defines a new burnPercent.
  **Visibility**
  external onlyOwner
  **Input parameters**
    - uint256 _newBurnPercent,
    - uint256 _newDivider

  **Constraints**
  None
  **Events emit**
  None
  **Output**
  None
- ***createStakingPool***
  **Description**
  Creates a new StakingPool contract.
  **Visibility**
  external
  **Input parameters**
    - address _stakingToken,
    - address _poolToken,
    - uint256 _startDate,
    - uint256 _finishDate,
    - uint256 _poolTokenAmount

  **Constraints**
    - Any user can create a StakingPool, but must provide a feeToken transfer fee.

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

- ***isContract***
  **Description**
  Checks is the address of the contract.
  **Visibility**
  private view.
  **Input parameters**
    - address _addr

  **Constraints**

None

**Events emit**

None

**Output**

Bool

## Audit overview

### 🟥 🟥 🟥 🟥 Critical

1. Function transferFrom is not checked for success and can return false value. Use SafeTransfer function instead.

Update: During remediation check issue was fixed.

### 🟨 🟨 🟨 High

No high severity issues found.

### 🟩 🟩 Medium

No medium severity issues found.

### 🟦 Low

No low severity issues found.

### 🟦 Lowest / Code style / Best Practice

1. Suboptimal memory usage. Staking as organized as in StakingPool contract uses two mappings.

   More convenient is to use accumulation pattern as in this article: Bogdan Batog, Lucian Boca, Nick Johnson, "Scalable Reward Distribution on the EthereumBlockchain" https://uploads-ssl.webflow.com/5ad71ffeb79acc67c8bcdaba/5ad8d1193a40977462982470_scalable-reward-distribution-paper.pdf. Like in other parts, rewardPerBlock needs to be calculated by balance.

Update: During remediation check issue was not fixed. This risk is acceptable.

2. StakingPool Constructor: poolTokenAmount parameter is not needed rewardPerBlock = rewardToken.balanceOf(address(this)).div(finishBlock.sub(lastRewardBlock));

Update: During remediation check issue was fixed.

3. getMultiplier: it can be made as internal function;

Update: During remediation check issue was fixed.

4. Functions stakeTokens and withdrawStake. Code duplication. Function stakeTokens has almost the same component as withdrawStake function.

Update: During remediation check issue was fixed.

## 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** high, **0** medium, **0** low and **5** lowest issues during the audit.

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

| Category | Check Item | Comments |
|---|---|---|
| Code review | ▪ Reentrancy | ▪ Lack of reentrancy guard checks. |
| | ▪ ERC20 API Violation | ▪ Transfer from method result success is ignored. |
| | ▪ Business Logics Review | ▪ Lack of whitepaper and documentation. |
| | ▪ Style guide violation | ▪ A lot of code-style issues were found. |

Update: Most contract’s vulnerabilities were fixed after the audit was done. One low lowest severities left in contract and this risk is acceptable. For details check “Audit overview” section.

# 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.