HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: UTU
Date: September 15th, 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 UTU
Type	Token Smart Contract
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Github repository	https://github.com/utu-protocol/utu-coin/tree/master/contracts
Commit number	273c027d8acb0d6b3ce94556a56f712e76246fe6
Deployment address	https://etherscan.io/address/0xa58a4f5c4bb043d2cc1e170613b74e767c94189b
Timeline	14th SEP 2020 - 15th SEP 2020
Changelog	15th SEP 2020 - Initial Audit 10th OCT 2020 - Add git address and commit to report

Contractor Contacts:

Role	Name	Email
Responsible Manager	Andrew Matiukhin	a.matiukhin@hacken.io

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

Table of contents

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

Introduction

Hacken OÜ (Consultant) was contracted by UTU (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 September 14th, 2020 - September 15th, 2020.

Scope

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

https://github.com/utu-protocol/utu-coin/tree/master/contracts	
Commit number 273C027D8ACB0D6B3CE94556A56F712E76246FE6	
./UTUToken.sol	In Scope of Review

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

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

Executive Summary

According to the assessment, the Customer's smart contract does not have high-level vulnerabilities and can be considered secure. It is recommended to fix one low-level issue.

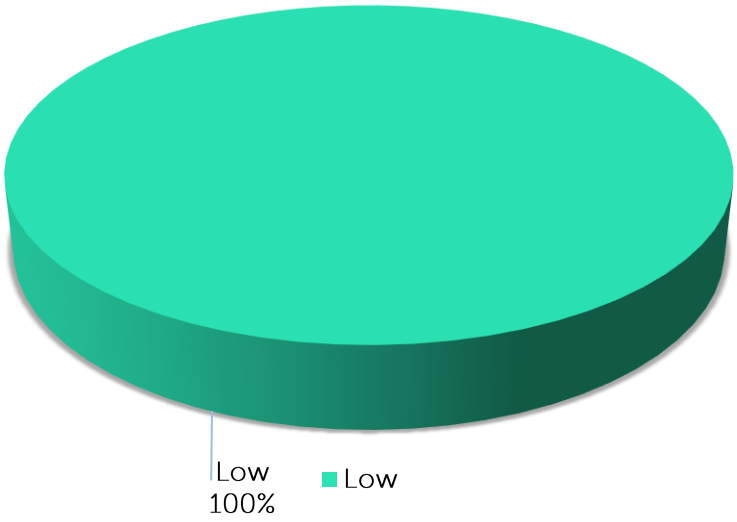
Insecure	Poor secured	Secured	We-secured

You are here

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 low severity issues during audit.

Graph 1. The distribution of vulnerabilities.



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

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.

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

AS-IS overview

UTUToken.sol

Description

UTUToken is a token contract with cap and custom roles managment.

Imports

UTUToken contract has 7 imports:

- *Ownable.sol* – from OpenZeppelin;
- *AccessControl.sol* – from OpenZeppelin;
- *ERC20.sol* – from OpenZeppelin;
- *ERC20Capped.sol* – from OpenZeppelin;
- *SafeERC20.sol* – from OpenZeppelin;
- *SafeMath.sol* – from OpenZeppelin;
- *Math.sol* – from OpenZeppelin;

Inheritance

UTUToken contract inherits *ERC20Capped*, *Ownable*, *AccessControl*.

Usings

UTUToken contract use:

- *SafeERC20* for *ERC20* type
- *SafeMath* for *uint256* type

Structs

UTUToken contract does not have data structures.

Fields

UTUToken contract has 6 fields:

- *bytes32 public constant MINTER_ROLE* – a minter role indicator;
- *bytes32 public constant BURNER_ROLE* – a burner role indicator;
- *bytes32 public constant RECOVERY_ROLE* – a recovery role indicator;

- *mapping(bytes32 => mapping(address => uint256)) public roleAssigned* – assigned roles;
- *uint256 public activationDelay* – delay before role activation;
- *bool public isMigrating* – if true then migration started;

Modifiers

UTUToken contract does not have data modifiers.

Functions

UTUToken has 9 functions:

- ***constructor***

  #### Description

  Initializes contract.

  #### Visibility

  public

  #### Input parameters

  - *uint256 _cap* – a cap;
  - *address[] memory _initialHolders* – a list of holders;
  - *uint256[] memory _initialBalances* – a list of balances;

  #### Constraints

  - The length of the arrays of holders and balances must be equal.

  #### Events emit

  None

  #### Output

  None

- ***setupMinter***

  #### Description

Assign a new minter.

Visibility

public

Input parameters

- *address _who* – an address of the new minter;

Constraints

- Only Owner can call it.

Events emit

None

Output

None

- *setupBurner*

Description

Assign a new burner.

Visibility

public

Input parameters

- *address _who* – an address of the new burner;

Constraints

- Only Owner can call it.

Events emit

None

Output

None

- *setupRecovery*

  Description

  Designates who can return ETH and tokens sent to this contract.

  Visibility

  public

  Input parameters

  - *address _who* – an address of the new recoverer;

  Constraints

  - Only Owner can call it.

  Events emit

  None

  Output

  None

- *mint*

  Description

  Mint new tokens and transfer them.

  Visibility

  public

  Input parameters

  - *address to* – a recipient of newly minted tokens;
  - *uint256 amount* – an amount of tokens to mint;

  Constraints

  - Cannot mint while migrating.
  - Only Minter can call it.
  - Minter must be active.

Events emit

None

Output

None

- *burn*

  Description

  Burn tokens belonging to the caller.

  Visibility

  public

  Input parameters

  - *uint256 amount* – an amount of tokens to burn;

  Constraints

  - Only Burner can call it.
  - Burner must be active.

  Events emit

  None

  Output

  None

- *startMigration*

  Description

  Starting the migration process means that no new tokens can be minted.

  Visibility

  public

  Input parameters

None

Constraints

- Only Owner can call it.

Events emit

None

Output

None

- *recoverTokens*

Description

Recovers tokens accidentally sent to a token contract.

Visibility

external

Input parameters

- *address _token* – an address of the token to be recovered. 0x0 address will recover ETH;
- *address payable _to* – a recipient of the recovered tokens;
- *uint256 _balance* – an amount of tokens to be recovered;

Constraints

- Only Recoverer can call it.
- Recoverer must be active.
- Recipient address can not be zero.

Events emit

None

Output

None

- *active*

Description

Checks if the msg.sender role has been assigned and the delay has passed.

Visibility

private view

Input parameters

- *bytes32 _role* – a role indicator;

Constraints

None

Events emit

None

Output

Returns true if the role is active, or false otherwise.

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

Audit overview

Critical

No critical severity issues were found.

High

No high severity issues were found.

Medium

No medium severity issues were found.

Low

1. Checking for the length of arrays does not guarantee data consistency in *constructor* input parameters. It is recommended to use an array of data structures instead of two arrays.

Lowest / Code style / Best Practice

No lowest severity issues were found.

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

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	■ Data Consistency	Checking the length of arrays does not guarantee consistency of data in the input parameters of the *constructor*. Because the order of the elements in the array could be out of order.

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

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

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.