HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Divs
Date: January 06th, 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 (45 pages).
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Token, Token sale
Platform	Tron / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review.
Repository	Private https://shasta.tronscan.org/#/contract/TJE8Mv2kbKsbZrJW4aFRBktzW93ZK673fR https://shasta.tronscan.org/#/contract/TA7Wxjfbpsip4Fdb7Yv1dGGkUAXXHe7cWN
Commit	https://shasta.tronscan.org/#/contract/TC5S6yRLCtfizCiwcrca7NnAHVK4GGvSqA
Deployed contract	https://shasta.tronscan.org/#/contract/TTFoHFyNhtTJZ6oX18EzAtwm3AZKajkizd https://shasta.tronscan.org/#/contract/TRscWDKqiHTMzfjdTAZZcMgMHiDUBeaK5W https://shasta.tronscan.org/#/contract/TRBGVKnYXLDySBnZnVy896xqbv8L2cXnzU https://shasta.tronscan.org/#/address/TALYz21hpGroczGV2mbsSNLyPVzR2xKhuL
Changelog	December 16th, 2020 – Initial Audit January 06th, 2021 – Remediation check

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 Divs (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 December 11th, 2020 – December 16th, 2020.

Remediation check was conducted January 06th, 2021.

Scope

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

Contract deployment address. Token on testnet:
https://shasta.tronscan.org/#/contract/TJE8Mv2kbKsbZrJW4aFRBktzW93ZK673fR
Presale Round 1:
https://shasta.tronscan.org/#/contract/TA7Wxjfbpsip4Fdb7Yv1dGGkUAXXHe7cWN
Presale Round 2:
https://shasta.tronscan.org/#/contract/TC5S6yRLCtfizCiwcrca7NnAHVK4GGvSqA
Presale Round 3:
https://shasta.tronscan.org/#/contract/TTFoHFyNhtTJZ6oX18EzAtwm3AZKajkizd
Presale Round 4:
https://shasta.tronscan.org/#/contract/TRscWDKqiHTMzfjdTAZZcMgMHiDUBeaK5W
Stake Data contract:
https://shasta.tronscan.org/#/contract/TRBGVKnYXLDySBnZnVy896xqbv8L2cXnzU
Stake Handler Contract:
https://shasta.tronscan.org/#/address/TALYz21hpGroczGV2mbsSNLyPVzR2xKhuL Files:
DivToken.sol
Round1Presale.sol
Round2Presale.sol
Round3Presale.sol
Round4Presale.sol
StakeDataTron.sol
StakeHandlerTron.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

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

	✦ Transaction-Ordering Dependence
	✦ Style guide violation
	✦ Costly Loop
	✦ TRC20 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
	✦ Assets integrity
	✦ User Balances manipulation
	✦ Kill-Switch Mechanism
	✦ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart has issues, but most of them were fixed before remediation check.

Insecure	Poor secured	Secured	Well secured

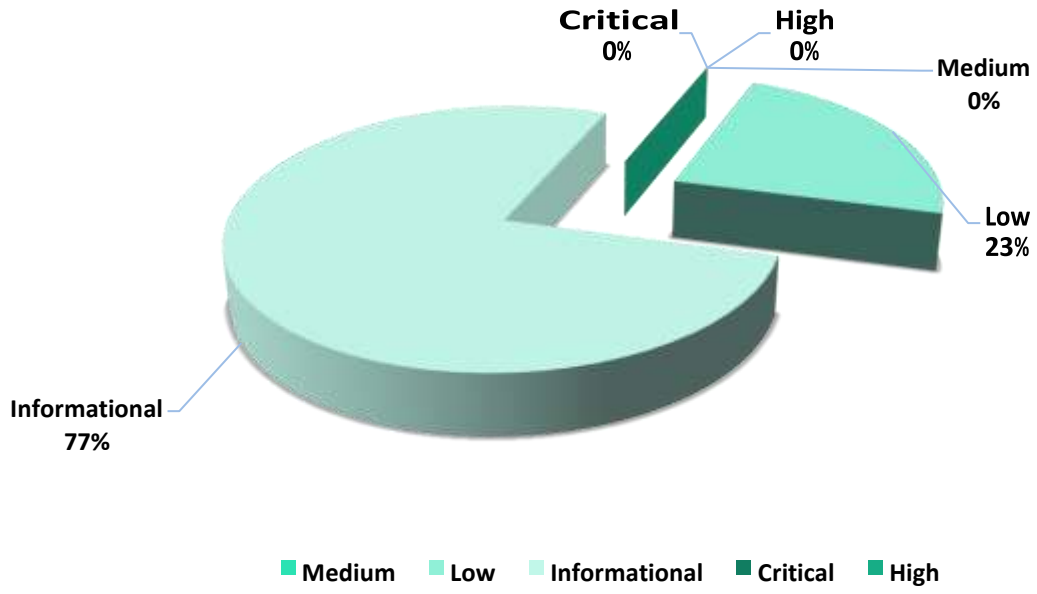
You are here

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 **0** critical **0** high, **0** medium, **3** low and **10** informational issues during remediation check.

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

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 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.

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

AS-IS overview

DivToken.sol

Description

DivToken is a contract used for the token crowdsale. **Imports**

DivToken contract hasn't the following imports: **Usages**

DivToken contract has the following custom usages:

- SafeMath for uint256

Structs

CrowdSale contract has no the following data structures:

Enums

CrowdSale contract has no custom enums.

Events

DivToken contract has the following events:

- event Transfer(address indexed from, address indexed to, uint256 value);
- event Approval(address indexed owner, address indexed spender, uint256 value);

Modifiers

DivToken has no custom modifiers.

Fields

DivToken contract has following constants:

- mapping (address => uint256) private balances;
- mapping (address => mapping (address => uint256)) private allowed;
- string public constant name = "Divs"; //Name
- string public constant symbol = "DIVS"; // Symbol

- uint8 public constant decimals = 8;

Functions

DivToken has following public functions:

- **totalSupply**
  Description
  Assigns the minting role.
  Visibility external view
  Input parameters
  None
  Constraints
  None
  Events emit
  None **Output**
  uint256
- **balanceOf**
  Description
  Assigns the

Audit overview

Critical

1. None

High

1. None

Medium

1. None

Low

1. StakeDataTron.sol: library Math - not used anywhere, and unnecessary in a contract.
2. StakeHandlerTron.sol: can simplify initialSupply: 'uint256 initialSupply = 100000000; uint256 public token_precision_multiplier = SafeMath.mul(initialSupply, 10**uint256(decimals));' to 'uint256 public token_precision_multiplier = SafeMath.mul(100000000, 10**uint256(decimals));'.

3. Round1Presale.sol, Round2Presale.sol, Round3Presale.sol and Round4Presale.sol: contracts are almost identical, the differences are only in the variables buyLimit and tokenPrice. The better way is to create a general class with all logic and add parameters to the constructor to calculate buyLimit and tokenPrice.

Lowest / Code style / Best Practice

1. StakeDataTron.sol: the function getMyStakeedTokens does the same as getStakedTokens.
2. StakeHandlerTron.sol: replace assert with require and add a more straightforward error description to require.
3. Round1Presale.sol, Round2Presale.sol, Round3Presale.sol and Round4Presale.sol: now the contracts' name is identical (StoredPresale) - rename them for convenience. Example: Round1Presale, Round2Presale, Round3Presale and Round4Presale.
4. Round1Presale.sol, Round2Presale.sol, Round3Presale.sol and Round4Presale.sol: CeoAddress and multiplier parameters be private as well.
5. Round1Presale.sol, Round2Presale.sol, Round3Presale.sol and Round4Presale.sol: function buy() - Replace assert with require.
6. DivToken.sol: add a more straightforward error description to require.
7. StakeHandlerTron.sol: many public functions, maybe some of them can be made private.
8. StakeDataTron.sol: stakeHandler address should be defined in constructor.
9. StakeHandlerTron.sol: the check can be transferred and used as modify - 'require (msg.sender == ceoAddress, "Unauthorized").
10. StakeDataTron.sol: the check can be transferred and used as modify - 'require (msg.sender == ceoAddress, "Unauthorized").

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

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.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

During remediation check (January 06th, 2021) security engineers found **0** critical **0** high, **0** medium, **3** low and **10** informational 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.
	✦ Style guide violation	✦ A lot of code-style issues were found.

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 a local 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.