HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: ThriveCoin, Inc.
Date: April 13th, 2022

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 ThriveCoin, Inc..
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type of Contracts	ERC20 token; Vesting
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Timeline	01.04.2022 - 13.04.2022
Changelog	07.04.2022 - Initial Review 13.04.2022 - Revision

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 ThriveCoin, Inc. (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

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

Repository:
https://github.com/ThriveCoin/thc-smart-contracts-sol
Commit:
db78d6b4cf6dcb9e49e7a1f146851f77fa872bbf
5b755d9d733e1b8df71d4d39faeb85983213c83c (revision)
Documentation: Yes
(https://github.com/ThriveCoin/thc-smart-contracts-sol/tree/main/test)
JS tests: Yes
(https://github.com/ThriveCoin/thc-smart-contracts-sol/tree/docs/docs)
Contracts:
ERC20Blockable.sol
ERC20DynamicCap.sol
ERC20LockedFunds.sol
ThriveCoinERC20Token.sol
ThriveCoinERC20TokenPolygon.sol
ThriveCoinVestingSchedule.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 ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ EIP standards violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency

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

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency ▪ Kill-Switch Mechanism

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

Executive Summary

The score measurements details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided perfect functional and technical requirements. The total Documentation Quality score is **10** out of **10**.

Code quality

The total CodeQuality score is **10** out of **10**. Code is written according to best practices. Test coverage is significant.

Architecture quality

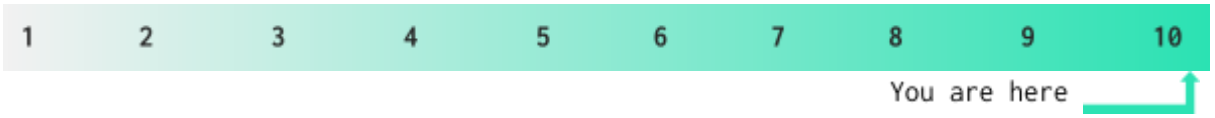
The architecture quality score is **10** out of **10**. Code is carefully divided by files. Each subcontract performs its specific role.

Security score

As a result of the audit, security engineers found **0** severity issues. The security score is **10** out of **10**. All found issues are displayed in the “Issues overview” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **10**.



Notices

1. The owner can block any account from transferring and receiving tokens.
2. The owner can revoke vesting or change beneficiary if such arguments were provided on creation.
3. The owner can pause all transfers on the contract.

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 cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution

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

Findings

Critical

No critical severity issues were found.

High

1. **The old owner keeps admin roles**

   Admin roles are not removed from the old owner on transferring ownership, so the address keeps some additional permissions.

   This leads to the old owner having a significant impact on the contract.

   Contract: ThriveCoinERC20Token.sol

   Function: transferOwnership

   Contract: ThriveCoinERC20TokenPolygon.sol

   Function: transferOwnership

   Recommendation: revoke the roles of the old owner.

   Status: Fixed (Revised Commit: 5b755d9)

2. **Missing revocable validation**

   According to the documentation, the function should be available to call only if *revocable_* parameter was set as *true* on contract deploying. However, now the validation is missed, and availability does not depend on this parameter.

   This may lead to loss of trust to the contract owner.

   Contract: ThriveCoinVestingSchedule.sol

   Function: revoke

   Recommendation: validate the parameter.

   Status: Fixed (Revised Commit: 5b755d9)

3. **Claiming logic corrupted**

   Users may not withdraw any tokens if deposited only part of the vested amount. *claim* function does not validate if the contract has enough tokens to satisfy withdrawal.

   This may lead to unavailability to claim tokens and loss of trust to the contract owner.

   Contract: ThriveCoinVestingSchedule.sol

   Function: claim

Recommendation: validate if the contract has enough funds; It is possible to implement a function that will indicate if the vesting is fulfilled with tokens.

Status: Fixed (Revised Commit: 5b755d9)

Medium

1. Depositing and withdrawing logic corrupted

Revoking vesting may block some tokens on the contract if there were deposited more tokens than the vested amount is. *revoke* function withdraws unclaimed part of vesting but leaves some possible assets on the contract.

Vesting may not be revoked if deposited only part of the vested amount. *revoke* function does not validate if the contract has enough tokens to satisfy withdrawal.

This may lead to loss of unexpectedly transferred tokens or to impossibility to revoke vesting at the right moment.

Contract: ThriveCoinVestingSchedule.sol

Function: revoke

Recommendation: transfer the whole balance of the contract on revoking; implement a safe-vesting function to send no more funds than needed.

Status: Fixed (Revised Commit: 5b755d9)

2. Public access to service function

It is a service function, and it should not be accessible by users. It is overridden in inheritant contracts, but it is better to declare it is internal and give an opportunity to wrap the function in different ways in inheritant contracts.

The issue could lead to unexpected behavior in the future use of this contract.

Contract: ERC20DynamicCap.sol

Function: updateCap

Recommendation: declare the function as *internal*.

Status: Fixed (Revised Commit: 5b755d9)

3. Locked tokens may not be spent

Even if tokens are locked for a spender, the spender may not transfer them without allowance.

This may lead to the accumulation of locked assets without the ability to spend them for the owner or the spender.

If it is a feature, documentation should provide information about how to allow spending locked assets.

Contract: ERC20LockedFunds.sol

Recommendation: make it clear how to use locked assets; maybe allowance logic may be changed to prevent locked assets with zero allowance.

Status: Fixed (Revised Commit: 5b755d9)

Low

1. Floating pragma

The contracts use floating pragma ^0.8.0.

Recommendation: consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed (Revised Commit: 5b755d9)

2. Functions that can be declared as external

In order to save Gas, public functions that are never called in the contract should be declared as *external*.

Contract: ERC20LockedFunds.sol

Functions: lockedBalanceOf, lockedBalancePerAccount, lockAmount, lockAmountFrom, unlockAmount

Contract: ThriveCoinERC20Token.sol

Functions: decimals, blockAccount, unblockAccount

Contract: ThriveCoinERC20TokenPolygon.sol

Functions: pause, unpause, decimals

Recommendation: aforementioned should be declared as *external*.

Status: Fixed (Revised Commit: 5b755d9)

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

Recommendations

1. Implement a function that gives an ability to take back money that exceeds the amount locked for vesting.

   Contracts: ThriveCoinVestingSchedule.sol

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 by the best industry practices at the date of this report, with 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 the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. 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 the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.