SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Thrupenny
Date: Sep 26rd, 2022

This report 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 can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Thrupenny
Approved By	Noah Jelich | Lead Solidity SC Auditor at Hacken OU
Type	Vesting
Platform	EVM
Network	Ethereum
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	http://Thrupenny.io
Timeline	13.09.2022 - 26.09.2022
Changelog	14.09.2022 - Initial Review 26.09.2022 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by **Thrupenny** (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:

Initial review scope

Repository:
https://bitbucket.org/exiotech/tpy-vesting/src/master/
Commit:
ceffed4f3c3735b630199d11fd67aa9a815ab370
Documentation:
Whitepaper

Technical description

Functional requirements

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/mocks/TestToken.sol
SHA3: 6676524a7c52f24b2de8ef500966ee19a8811893834351ae259827565b20f163

File: ./contracts/mocks/TPYToken.sol
SHA3: 4327cfb0776ecbf17888bad3131e0204d0f194cfdf4b1dce51aa3db322bb5628

File: ./contracts/mocks/VestingMock.sol
SHA3: 82473a59dc0d14fcd615e7adb3cdcbe6bd5f26baeaeae146fa98563e28f67a35

File: ./contracts/Vesting.sol
SHA3: d13aced01e37d69bacf492635e25f237a1a3d26d10f492facac58b27845c08c8

Second review scope

Repository:
https://bitbucket.org/exiotech/tpy-vesting/src/master/
Commit:
f0a5107ffc8b0945aa798ebea1867feabe68b264
Documentation:
Whitepaper

Technical description

Functional requirements

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/mocks/TestToken.sol
SHA3: 9073f8235998fd7e5d03c352a066ee0145b41fb76becf4138917a01084b7cf64

File: ./contracts/mocks/TPYToken.sol

```
SHA3: ff7cb5319991faf6e75eea13d1e7cf82f47cf5398629917c2f43a67c79a6592b

File: ./contracts/mocks/VestingMock.sol
SHA3: b59bdb22adb8a6d3e9d70da60d87d15f649b483ef40ccc92c8eeedf0a3a22f0e

File: ./contracts/Vesting.sol
SHA3: 3a47be316036b51cec805676c9d5c2b8168b81d9605feb8c5a27ebf459f537c1
```

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.

Executive Summary

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

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is well configured.

Test coverage

Test coverage of the project is **98.72%**.

- Deployment and basic user interactions are covered with tests.
- Negative and positive cases are covered with tests.
- Interactions by several users are tested.

Security score

As a result of the audit, the code contains **2** low severity issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

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

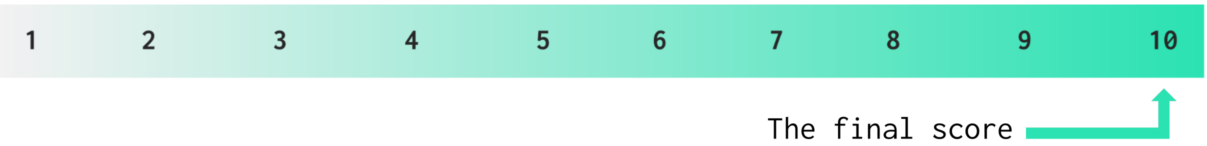

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
14 September 2022	3	1	1	0
23 September 2022	2	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Not Relevant
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Not Relevant
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Failed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
Assets integrity	Custom	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	Custom	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	Custom	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	Custom	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	Custom	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Not Relevant

Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

TPY Vesting is an ERC20 Vesting system with the Vesting contract:

- *Vesting* — the contract that gives certain wallet addresses access to claim a certain percentage of the funds according to the timeline and the rules. There are two rules defined by the owner:
  - have a list of wallet addresses and addresses that will receive the tokens allocated to them according to the same schedule.
  - have provided 1 wallet address, which is the multi-sig wallet of the team (made through Gnosis Safe). The wallet will receive the token distribution with a different schedule than the first category of token recipients.

Privileged roles

- The owner of the *Vesting* contract can:
  - withdraw mis-sent tokens from the contract.
  - create vesting schedules.

Findings

🟥🟥🟥🟥 Critical

No critical severity issues were found.

🟧🟧🟧 High

1. Access control violation

The owner of the Vesting contract can delete any vesting schedule and withdraw its fund with *emergencyWithdraw* function, and move any vest ownership to another address using *updateTarget* function.

Path: ./contracts/Vesting.sol: emergencyWithdraw(), updateTarget()

Recommendation: Remove emergencyWithdraw function and restrict owner privilege in the updateTarget function.

Status: Fixed (f0a5107ffc8b0945aa798ebea1867feabe68b264)

🟨🟨 Medium

1. Redundant SafeERC20 implementation

SafeERC20 is necessary in case a transferred token is not in ERC20 standards. Since the token used in Vesting is the TPY token and is in ERC20 standards, the usage of SafeERC20 is redundant.

Usage of SafeERC20 increases Gas costs both during deployment and each safe transfer transaction.

Path: ./contracts/Vesting.sol

Recommendation: Remove SafeERC20 implementation from Vesting and check the return value of IERC20's transfer method.

Status: Fixed (f0a5107ffc8b0945aa798ebea1867feabe68b264)

🟦 Low

1. Floating pragma

The project uses floating pragma ^0.8.7.

Path: all

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

Status: Fixed (f0a5107ffc8b0945aa798ebea1867feabe68b264)

2. State variables can be declared immutable

Variable token value is set in the constructor. This variable can be declared immutable.

This will lower the Gas taxes.

Path: ./contracts/Vesting.sol

Recommendation: Declare mentioned variables as immutable.

Status: Fixed (f0a5107ffc8b0945aa798ebea1867feabe68b264)

3. **Style guides violation**

   The provided projects should follow the official guidelines.

   Path: ./contracts/Vesting.sol

   Recommendation: Follow the official Solidity guidelines.

   Status: Fixed (f0a5107ffc8b0945aa798ebea1867feabe68b264)

4. **Unused event**

   The EmergencyWithdrawal event is never used.

   Path: ./contracts/Vesting.sol

   Recommendation: Remove mentioned event.

   Status: New

5. **Variables that should be declared constant**

   State variables that do not change their (percentsPerStages and stagePeriods) value should be declared constant to save Gas.

   Path: ./contracts/Vesting.sol

   Recommendation: Declare the above-mentioned variables as constants.

   Status: New

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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

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, Consultant cannot guarantee the explicit security of the audited smart contracts.