HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Venom
Date: 28 Aug, 2023

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 Venom
Approved By	Oleksii Zaiats | SC Audits Head at Hacken OÜ
Type	Vesting
Platform	Venom
Language	TON-Solidity
Methodology	Link
Website	web3.world/vesting
Changelog	11.08.2023 - Initial Review 28.08.2023 - Second Review

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 Venom (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.

System Overview

Venom Vesting – a smart contract system allowing vesting contracts creation and management.

In-scope contracts:

- *IndexFactory* & *Index* – contracts that provide an ability to list decentralized data.
- *VestingFactory* (inherit *IndexFactory*) – contract allows *Vesting* and *NativeVesting* contracts deployment. It deploys *Index* contracts for vesting creator, recipient, and TIP-3 token.
- *NativeVesting* – vesting contract for native tokens. Should be fulfilled with direct deposit.
- *Vesting* – vesting contract for TIP-3 tokens. Should be fulfilled with a callback of the TIP-3 deposit.

Roles

- Only the vesting recipient specified on deployment is able to withdraw vested funds.
- Any user is able to create and fulfill vesting.

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

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**.

- A description of system functionality is provided.
- The technical description is comprehensive.

Code quality

The total Code Quality score is **9** out of **10**.

- Magic numbers usage is found.
- Style guide violations are found.
- The development environment is configured.

Test coverage

The coverage of the project with tests is about **85%**.

- Some minor view functions are not tested thoroughly. Check if all the implemented functions are called and the return values are checked.
- Missing access control tests for callbacks.

Security score

As a result of the audit, the code does not contain security issues. The security score is **10** out of **10**.

All found issues are displayed in the Findings section of the report.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.3**.

The system users should acknowledge all the risks summed up in the Risks section of the report.

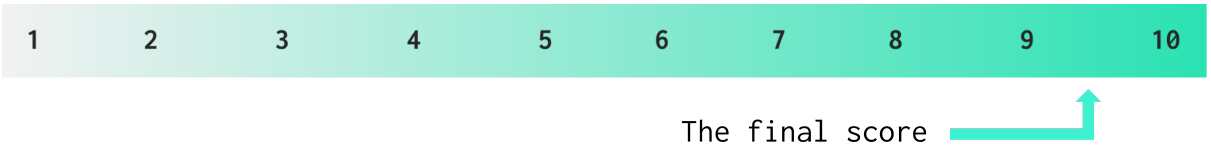

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
11 August 2023	2	1	0	1
28 August 2023	0	0	0	0

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

Risks

- Depending on TIP-3 token implementation funds may get stuck in case not enough value is attached to the call (especially during performing a deposit to the *Vesting* contract).
- In case of the Gas price change, the minimal attached amount may be not enough for performing the execution of the whole call chain.
- Users may be unable to retrieve the list of their vestments on-chain.
- In case the vesting period is overlong, vesting contracts may need additional funding to not be frozen.

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

Checked Items

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

Item	Description	Status
Integer Overflow and Underflow	All math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	It is recommended to use a recent version of the TON-Solidity compiler.	Passed
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Block values as a proxy for time	Block numbers should not be used for time calculations.	Not Relevant
Signature Reuse	Signed messages that represent an approval of an action should not be reusable.	Not Relevant
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	The code should not contain unused variables until they are not justified by design.	Passed
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	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

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

Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Not Relevant
Gas Management	Transaction execution costs should not depend dramatically on the amount of data stored in the contract. Contracts should validate that incoming value is enough to perform the whole call chain.	Passed
Compiler Warnings	The code should not force the compiler to throw warnings.	Passed
Style Guide Violation	Style guides and best practices should be followed.	Failed (I05)
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	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	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
Test Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. The usage of contracts by multiple users should be tested.	Failed (Code Coverage)
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed

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

Findings

Critical

C01. Transaction Replay Attack

Impact	High
Likelihood	High

The *sendTransaction* function of the *@broxus/contracts/contracts/wallets/Account.tsol* wallet implementation is considered to be vulnerable to a transaction replay attack due to *tvm.accept()* call being performed before all essential checks are done and an arbitrary *flag* for a performed call could be provided.

The *dest.transfer(..)* call may fail, for, example, due to a low contract balance. The *flag* value is not checked for containing *+2 (ignore exceptions)* modifier. Thus, the failing transaction could be replayed by a validator draining the wallet balance.

The *pragma AbiHeader expire* is used, however, it does not help due to validator is not bounded by the number of times replaying the transaction in one block.

Path: ./contracts/Wallet.sol

Recommendation: Use trusted wallet implementations, validate that contract will not fail after a *tvm.accept()* call, or remove the file.

Valuable link: everscale.guide/smart_contracts/replay_protection

Found in: ce6dee4

Status: Fixed (Revised commit: 46a4f7d)

High

No high severity issues are found.

Medium

M01. Inconsistent Gas Management

Impact	Medium
Likelihood	Medium

Although the Gas price is stable for now, it may be changed in the future.

The variables and constants store needed value approximation, not the Gas amount.

Paths:

- ./contracts/indexer/IndexFactory.tsol: _indexDeployValue, _indexDestroyValue
- ./contracts/indexer/NativeVesting.sol: MIN_MSG_VALUE
- ./contracts/indexer/Vesting.sol: TOKEN_WALLET_DEPLOY_VALUE, MIN_MSG_VALUE
- ./contracts/indexer/VestingFactory.sol: VESTING_DEPLOY_VALUE

Recommendation: Store the minimal amount of Gas attached to the call, not the minimal value, and check if the provided value is enough using the *gasToValue(..)* function.

Found in: ce6dee4

Status: Mitigated (The *gasToValue(..)* functionality is still experimental)

Low

L01. Floating Pragma

Impact	Low
Likelihood	Medium

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Paths: ./contracts/*

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

Note: Locking pragma to *^0.Y.Z* is considered to be valid as allows applying patches without breaking changes.

Found in: ce6dee4

Status: Fixed (Revised commit: 46a4f7d)

L02. Outdated Compiler Version

Impact	Low
Likelihood	Medium

The *0.62.0* version of the TON-Solidity Compiler is used, however, it is considered to be outdated. Newer compiler versions contain fixes for various issues.

Paths: ./contracts/*

Recommendation: Use the relevant compiler version.

Note: updating pragma over *0.67.0* requires the following changes:

- *constructor* should not have visibility specifier
- *now* should be replaced with *block.timestamp*
- *<TvmSlice>.decode* should be replaced with *<TvmSlice>.load*

Found in: ce6dee4

Status: Mitigated (The *0.62.0* version has no disclosed bugs related to the code)

Informational

I01. Code Duplication

- *tokens_to_claim*/*value_to_claim* value calculation is duplicated in *pendingVested()* and *claim()* functions.

  Paths: ./contracts/Vesting.sol, ./contracts/NativeVesting.sol

- The *Vesting* and *NativeVesting* contracts have common logic that could be moved to a base contract.

  Paths: ./contracts/Vesting.sol, ./contracts/NativeVesting.sol

- The common sanity checks can be compiled into a modifier.

  Path: ./contracts/VestingFactory.sol: deployNativeVesting(), deployVesting()

Recommendation: Extract common logic into a standalone piece of code.

Found in: ce6dee4

Status: Reported

I02. Redundant Branching

There is an unnecessary code branching present. It can be omitted by extracting each piece of logic into a separate *internal* function, only used where it fits.

Path: ./contracts/VestingFactory.sol: onVestingDeployed()

Recommendation: Logical branching should be split into separate functions and common logic - kept in one function.

Found in: ce6dee4

Status: Reported

I03. Magic Numbers Usage

Constant number values are used as message flags that obscure their purpose.

Paths:

- ./contracts/indexer/Index.tsol
- ./contracts/indexer/IndexFactory.tsol

Recommendation: Use *MsgFlag* library to make flags declarative.

Found in: ce6dee4

Status: Reported

I04. Implicit Call Flag

Some messages are implicitly configured. The *flag* field is missing.

Paths:

- ./contracts/Vesting.tsol: _setupTokenWallets()
- ./contracts/indexer/IndexFactory.tsol: destructIndex()

Recommendation: Consider specifying *flag* to make call configuration explicit.

Found in: ce6dee4

Status: Fixed (Revised commit: 46a4f7d)

I05. Style Guide Violation

There are variable names defined in the snake case.

Paths:

- ./contracts/interfaces/IFactory.tsol: vesting_amount, vesting_start, vesting_end
- ./contracts/NativeVesting.tsol: remaining_amount, send_gas_to, value_to_claim, period_left, period_passed,
- ./contracts/Vesting.tsol: remaining_amount, tokens_to_claim, period_left, period_passed,
- ./contracts/VestingFactory.tsol: deploy_nonce, vestings_deployed, vesting_amount, vesting_start, vesting_end, vesting_contract_type, vesting_address

Recommendation: Follow the style guides, use camel case for variable and parameter names.

Found in: ce6dee4

Status: Reported

I06. Incorrect File Extension

It is considered best practice usage of *tsol* extension for TON-Solidity files.

Paths: ./contracts/*.sol

Recommendation: Convert *sol* extension into *tsol*.

Found in: ce6dee4

Status: Reported

I07-1. Redundancies

The return parameter names are redundant in the functions due to explicit return statements being used.

Paths:

- ./contracts/Vesting.sol: getDetails()
- ./contracts/NativeVesting.sol: getDetails()

Recommendation: Eliminate mentioned redundancies.

Found in: ce6dee4

Status: Mitigated (The return parameter names provide ABI of the functions return data)

I07-2. Redundancies

The else branch is redundant in the functions.

Paths:

- ./contracts/Vesting.sol: pendingVested()
- ./contracts/NativeVesting.sol: pendingVested()

Recommendation: Eliminate mentioned redundancies.

Found in: ce6dee4

Status: Fixed (Revised commit: 46a4f7d)

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 based on best industry practices at the time of the writing 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 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, the Consultant cannot guarantee the explicit security of the audited smart contracts.

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

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	**Medium**
Medium Likelihood	**High**	**Medium**	**Low**
Low Likelihood	**Medium**	**Low**	**Low**

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

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

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

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

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/venom-blockchain/vesting
Commit	ce6dee47d0140e298b330089eb0fd00290d32a4d
Requirements	README.md
Contracts	File: ./contracts/indexer/IndexFactory.tsol SHA3: be263ac32451c56f9f88ff6fb9008a8932ac847119ffb37d8348c6786b6e07ee File: ./contracts/indexer/Index.tsol SHA3: fbc0098235d5d6cc77780d7331921f66d2638032a6e3251647198bbb9e781413 File: ./contracts/interfaces/IFactory.sol SHA3: 0ab9a8d0f0fdf74532d17f98675192f5ea2784e6c42dd9513b75bddc4292b805 File: ./contracts/NativeVesting.sol SHA3: 7ada960b9724f9a196bacb1598fd63e6b59552ff8faaee0e1ab251d13459cef0 File: ./contracts/VestingFactory.sol SHA3: d8d31cf6131ecc24238e1c174502e9c392770eb80e6d180abb7f2df05e596786 File: ./contracts/Vesting.sol SHA3: 3f0e97b2cf78e2727bc8efeaa65765b486c868cddfafdb31bae77d0369c95911 File: ./contracts/Wallet.sol SHA3: 276184a9a9d9090832533e3a386fc047889ab81ce9dd40a167757b83e36fd691

Second review scope

Repository	https://github.com/venom-blockchain/vesting
Commit	871af51db9c58c8054cb144aea0d4c2f368f692b
Requirements	README.md
Contracts	File: ./contracts/indexer/IndexFactory.tsol SHA3: 4daec32981c46e6dcbb06c886631f5d077e33eec5bd559a62794ab82d6d28687 File: ./contracts/indexer/Index.tsol SHA3: 5af1d74f5efdf16a86eaf1bd7754896639f0e5839dc4d6cc8d9203be3e95965d File: ./contracts/Vesting.sol SHA3: a6da90077d2e41b80fb3ea4c92656c6bb50c6feb8a303522d16c0ecd49741034 File: ./contracts/NativeVesting.sol SHA3: 5769e063dd3767d12a2db23004ca99e80c9f72df473eb8d753569f06aa64690c File: ./contracts/VestingFactory.sol SHA3: d8d31cf6131ecc24238e1c174502e9c392770eb80e6d180abb7f2df05e596786 File: ./contracts/interfaces/IFactory.sol SHA3: 0ab9a8d0f0fdf74532d17f98675192f5ea2784e6c42dd9513b75bddc4292b805