HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: ROWA Games
Date: 18 July, 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 ROWA Games
Approved By	Noah Jelich | Lead Solidity SC Auditor at Hacken OU
Type	ERC20 token; Vesting;
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://rowa.games/
Changelog	04.05.2023 - Initial Review 15.06.2023 - Second Review 18.07.2023 - Third Review

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

Table of contents

Introduction **4**
System Overview **4**
Executive Summary **5**
Risks **7**
Checked Items **8**
Findings **10**
Critical 10
C01. Requirements Violation 10
C02. Compilation Error 10
High 11
H01. Requirement Violation 11
H02. Requirement Violation 11
H03. Invalid Calculations; Requirement Violation 11
Medium 12
M01. Lock of Native Tokens 12
M02. Invalid Calculations 12
M03. Redundant Safemath Library 13
M04. Hash Collision 13
Low 14
L01. Functions That Can Be Declared External 1
L02. Redundant Code 1
L03. Missing Events 14
L04. Immutable Variables 15
L05. Constant Variables 15
Informational 16
I01. Style Guide Violation 16
I02. Boolean Equality 16
I03. Unnecessary Payable Variable 16
I04. Floating Pragma 16
I05. Redundant Virtual Keyword 17
I06. Redundant Token Transfer 17
I07. Unnecessary Variable Declaration 17
I08. Unused Identifier 18
Disclaimers **19**
Appendix 1. Severity Definitions **20**
Risk Levels 20
Impact Levels 21
Likelihood Levels 21
Informational 21
Appendix 2. Scope **22**

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

Introduction

Hacken OÜ (Consultant) was contracted by ROWA Games (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

ROWA Games - is a staking protocol with the following contracts:

- *RowaToken* – ERC-20 ownable, pausable token that mints all initial supply to a deployer. Additional minting is not allowed. Owner can start vesting through a contract.
  It has the following attributes:
  - Name: ROWA Token
  - Symbol: ROWA
  - Decimals: 5
  - Total supply: 1b tokens.
- *RowaVesting* – a contract that implements vesting mechanisms for different types of users.

Privileged roles

- The owner of the *RowaVesting* smart contract can start and stop vesting, and release tokens for any user.
- The owner of the *RowaTokens* smart contract can use pausable and snapshot mechanisms. Also it has the possibility to start vesting by transferring all tokens supply to the vesting smart contract.

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**.

- Functional requirements are partially missing.
  - Functional requirements related to the tokenomic are provided.
  - The technical requirements outlining how the system should operate and the necessary components for the smart contract are provided.
- Technical description is robust.
  - Inline comments to describe code behavior have been added.
  - NatSpec is provided: smart contract description, param description, return values description, validation rules.
  - Run instructions are provided.
  - Technical specification is provided.

Code quality

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

- Code contains no style issues.

Test coverage

Code coverage of the project is **99.04%** (branch coverage), with a mutation score of **49%**.

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is present.
- Interactions with several users are tested.
- The tests share similar inaccuracies with the contract logic, resulting in undetected issues.
- The comments within these tests do not accurately represent the executed code.

Security score

As a result of the audit, the code contains **no** security issues. The security score is **10** out of **10**.

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

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

Summary

According to the assessment, the Customer's smart contract has the following score: **9.9**. The system users should acknowledge all the risks summed up in the risks section of the report.

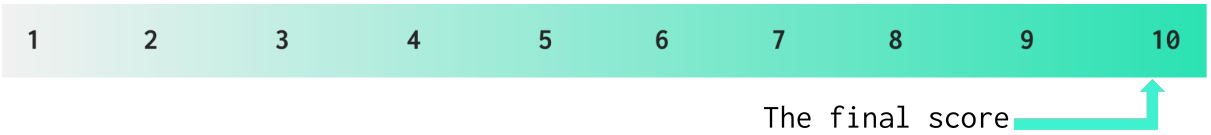

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
5 May 2023	5	4	1	2
13 June 2023	1	1	2	0
18 July 2023	0	0	0	0

Risks

- **Vestings for Team, Advisors, Partnerships can be revoked.**

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	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Not Relevant	
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	
SELFDESTRUCT Instruction	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

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

Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Not Relevant	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Passed	
Signature Unique Id	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. EIP-712 should be followed during a signer verification.	Not Relevant	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed	
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 if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Not Relevant	
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	
Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed	
Gas Limit and Loops	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.	Not Relevant	
Style Guide Violation	Style guides and best practices should be followed.	Passed	
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	
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed	
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. Requirements Violation

Impact	High
Likelihood	High

Functions *createPrivateSaleVesting()*, *createSeedSaleVesting()* create a vesting with cliff time according to the tokenomics documentation provided by the client. Moreover, according to the documentation vestings for advisors and team do not have a cliff time, but in the *createTeamVesting()*, *createAdvisorVesting()* functions, vestings are created with cliff times, and these times were taken from the initial *unlock(month)* column(tokenomics documentation). Initial *unlock(month)* column data is not used in any other functions, which create a vesting.

This causes the initial unlock and cliff timings described in the documentation to not match the code.

Path: ./RowaVesting.sol : startVGPVesting(), startLPVesting(), startLiqVesting(), startReserveVesting(), createPublicSaleVesting(), createPrivateSaleVesting(), createSeedSaleVesting(), createTeamVesting(), createAdvisorVesting(), createPartnershipVesting()

Recommendation: Adjust the documentation to reflect the code or adjust the code to reflect the documentation.

Found in: 2ecddd50

Status: Fixed (Revised commit: 23b3bb5)

C02. Compilation Error

Impact	High
Likelihood	High

A trivial error in the NatSpec prevents the RowaVesting contract from being compiled: *@param revokable_* in *createSeedSaleVesting()* function.

Path: ./RowaVesting.sol : createSeedSaleVesting()

Recommendation: Remove redundant NatSpec.

Found in: 2ecddd50

Status: Fixed (Revised commit: 23b3bb5)

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

■ ■ ■ High

H01. Requirement Violation

Impact	Medium
Likelihood	High

The function _computeReleasableAmount() allows for the release of initial funds before the vesting schedule start.

The second if block should be before the first one.

Path: ./RowaVesting.sol : _computeReleasableAmount()

Recommendation: Move the second if block.

Found in: 2ecddd50

Status: Fixed (Revised commit: 23b3bb5)

H02. Requirement Violation

Impact	Medium
Likelihood	High

The cliff period is documented and set for some funds, but is not used. In this implementation, initial vested amounts are distributed just after the cliff period passed.

This is contrary to the cliff mechanism, in which funds for withdrawal are available at the start of vesting, and only after its ending, vesting begins.

Path: ./RowaVesting.sol : _computeReleasableAmount()

Recommendation: Implement the missing feature.

Found in: 23b3bb5

Status: Fixed (Revised commit: 6e75228)

H03. Invalid Calculations; Requirement Violation

Impact	High
Likelihood	High

The function *_computeReleasableAmount()* computes the wrong vested amount. On line 814 the *vestedAmount* variable (an intermediate value) is proportionally computed over *vestingSchedule.amountTotal*, while it should be computed over *(vestingSchedule.amountTotal -*

initialAmount), because n line 819 the amount initially added to *vestedAmount* results in a double summation of its value.

The initial amount used in the vesting computation leads to a premature vesting of the tokens. The final vested tokens will not be more than the total amount.

Path: ./RowaVesting.sol : createSeedSaleVesting()

Recommendation: Remove adding *amountInitial* to *vestedAmount* variable at the 819 line.

Found in: 23b3bb5

Status: Fixed (Revised commit: 6e75228)

Medium

M01. Lock of Native Tokens

Impact	Medium
Likelihood	Medium

The contracts accept native tokens by the *receive()* and *fallback()* functions, but there are no mechanisms for withdrawals.

This can cause native coins to be locked in the contracts. Also, this results in an additional fee when deploying a smart contract.

Path: ./RowaVesting.sol : receive(), fallback()

Recommendation: Remove *receive()*, *fallback()* functions.

Found in: 2ecddd50

Status: Fixed (Revised commit: 23b3bb5)

M02. Invalid Calculations

Impact	Medium
Likelihood	Medium

The function *_computeReleasableAmount()* should calculate the accurate amount of tokens, which is available for release according to the vesting schedule. At the start of the function code (lines: *466* - *471*):

```
if (vestingSchedule.amountReleased < vestingSchedule.amountInitial)
{
        return vestingSchedule.amountInitial.
                sub(vestingSchedule.amountReleased);
}
```

The *if* statement compares the amount of released tokens and the initial amount of vested tokens. This statement will return a *true* value until releasing all amounts of initial supply.

This leads to returning the wrong value, because execution of the function would end inside this if statement even after the vesting periods end.

Path: ./RowaVesting.sol : _computeReleasableAmount()

Recommendation: Instead of returning a value, sum up the total number of tokens available for release at each stage of the vesting.

Found in: 2ecddd50

Status: Fixed (Revised commit: 6e75228)

M03. Redundant Safemath Library

Impact	Low
Likelihood	High

Starting with Solidity ^0.8.0, SafeMath functions are built-in. In such a way, the library is redundant.

Path: ./RowaVesting.sol

Recommendation: Remove the redundant library to save Gas on tx executions.

Found in: 2ecddd50

Status: Fixed (Revised commit: 23b3bb5)

M04. Hash Collision

Impact	High
Likelihood	Low

The vesting schedule ID is generated by hashing the holder and the vesting index.

Since these two variables are packed, the outcome is ambiguous and allows for collisions.

Path: ./RowaVesting.sol : computeVestingScheduleIdForAddressAndIndex()

Recommendation: Use abi.encode() instead of abi.encodePacked()

Found in: 2ecddd50

Status: Fixed (Revised commit: 23b3bb5)

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

Low

L01. Functions That Can Be Declared External

Impact	Low
Likelihood	Low

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

Paths: ./RowaVesting.sol : revoke(), computeReleasableAmount(), getLastVestingScheduleForHolder(), startVGPVesting(), startLPVesting(), startLiqVesting(), startReserveVesting()

./RowaToken.sol : snapshot(), pause(), unpause()

Recommendation: Use the external attribute for functions never called from the contract.

Found in: 2ecddd50

Status: Fixed (Revised commit: 23b3bb5)

L02. Redundant Code

Impact	Low
Likelihood	Medium

The total supply is enforced in _beforeTokenTransfer(), but this is not necessary as the contract does not implement external minting functions.

The if block can be removed, saving Gas on transfers.

Path: ./RowaToken.sol : _beforeTokenTransfer()

Recommendation: Remove the if block.

Found in: 2ecddd5

Status: Fixed (Revised commit: 23b3bb5)

L03. Missing Events

Impact	Low
Likelihood	Medium

Events for critical state changes should be emitted for tracking things off-chain.

Paths: ./RowaVesting.sol : revoke(), release(), startVGPVesting(), startLPVesting(), startLiqVesting(), startReserveVesting(),

createPublicSaleVesting(), createPrivateSaleVesting(), createSeedSaleVesting(), createTeamVesting(), createAdvisorVesting(), createPartnershipVesting()

./RowaToken.sol : snapshot(), pause(), unpause()

Recommendation: Create and emit related events.

Found in: 2ecddd50

Status: Fixed (Revised commit: 23b3bb5)

L04. Immutable Variables

Impact	Low
Likelihood	Medium

Multiple variables can be declared immutable to save gas on tx executions.

Path: ./RowaVesting.sol : _VGP_FUND, _LP_FUND, _LIQ_FUND, _RESERVE_FUND

Recommendation: Declare the mentioned variables as immutable.

Found in: 2ecddd5

Status: Fixed (Revised commit: 23b3bb5)

L05. Constant Variables

Impact	Low
Likelihood	Medium

Multiple variables can be declared constant to save Gas on tx executions.

Path: ./RowaVesting.sol : VGP_INITIAL_VESTING_PERCENTAGE, LP_INITIAL_VESTING_PERCENTAGE, PS_INITIAL_VESTING_PERCENTAGE, PRIVS_INITIAL_VESTING_PERCENTAGE, SEEDS_INITIAL_VESTING_PERCENTAGE, RESERVE_INITIAL_VESTING_PERCENTAGE, TEAM_INITIAL_VESTING_PERCENTAGE, ADVISORS_INITIAL_VESTING_PERCENTAGE, PARTNERSHIPS_INITIAL_VESTING_PERCENTAGE

Recommendation: Declare the mentioned variables as immutable.

Found in: 2ecddd5

Status: Fixed (Revised commit: 6e75228)

Informational

I01. Style Guide Violation

The provided projects should follow the official guidelines. There is an order of function violation.

Path: ./RowaVesting.sol;

Recommendation: Follow the official Solidity guidelines.

Found in: 2ecddd50

Status: Fixed (Revised commit: 23b3bb5)

I02. Boolean Equality

Boolean constants can be used directly and do not need to be compared to true or false.

Path: ./RowaVesting.sol : onlyInitialized(), onlyActive(), _computeReleasableAmount()

Recommendation: Remove boolean equality.

Found in: 2ecddd50

Status: Fixed (Revised commit: 23b3bb5)

I03. Unnecessary Payable Variable

The release() function uses address payable variables for transferring ERC20 tokens.

This is unnecessary code and leads to additional transaction gas costs.

Path: ./RowaVesting.sol : release()

Recommendation: Remove an unnecessary address payable variable.

Found in: 2ecddd50

Status: Fixed (Revised commit: 23b3bb5)

I04. Floating Pragma

The project uses floating pragmas `^0.8.19`.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Paths: ./RowaVesting.sol;

./RowaToken.sol;

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Found in: 2ecddd50

Status: Fixed (Revised commit: 23b3bb5)

I05. Redundant Virtual Keyword

The function getCurrentTime() is declared virtual, but since this contract is not meant to be inherited it can be removed.

Since this function is only returning block.timestamp, replacing its invocation with block.timestamp would maintain code readability while saving some gas on tx execution.

Path: ./RowaVesting.sol : getCurrentTime()

Recommendation: Remove the redundant virtual keyword.

Found in: 2ecddd5

Status: Fixed (Revised commit: 23b3bb5)

I06. Redundant Token Transfer

The RowaToken contract mints the initial supply to the deployer in the constructor, and then in the startVesting() function it transfers this sum from the deployer to the vesting contract.

It would be sufficient to directly mint the total supply to the vesting contract in the startVesting() function, to have a cleaner and more Gas efficient code.

Path: ./RowaToken.sol

Recommendation: Mint the total supply to the vesting contract in the startVesting() function.

Found in: 2ecddd5

Status: Fixed (Revised commit: 23b3bb5)

I07. Unnecessary Variable Declaration

These functions use the *currentTime* variable inside for storing the block.timestamp value. The *block.timestamp* value can be directly used in the *_createVestingSchedule()* function.

This leads to overuse of Gas in these functions.

Paths: ./RowaToken.sol : StartVGPVesting(), startLPVesting(), startLiqVesting(), startReserveVesting(), createPublicSaleVesting(), createPrivateSaleVesting(), createSeedSaleVesting(),

createTeamVesting(), createAdvisorVesting(), createPartnershipsVesting();

Recommendation: Use *block.timestamp* directly.

Found in: 23b3bb5

Status: Fixed (Revised commit: 6e75228)

I08. Unused Identifier

The structure *VestingSchedule* contains a *cliff* field which is never used in the code .

Path: ./RowaVesting.sol : VestingSchedule;

Recommendation: Remove redundant struct field.

Found in: 23b3bb5

Status: Fixed (Revised commit: 6e75228)

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/rowagames/ROWA-Token
Commit	2ecddd50
Whitepaper	https://docs.rowa.games/rowa/
Requirements	https://docs.rowa.games/rowa/
Technical Requirements	-
Contracts	File: ./RowaToken.sol SHA3: 0b4b386bc79edad9e50d2085a3c1fe1c1e9b22b899e6ad55c2849e5fbdba6ab9 File: ./RowaVesting.sol SHA3: b0b4eee65fe10ea2695fc7ef5f10b90d70c03eefe5344cf592c8fb6943a4d77f

Second review scope

Repository	https://github.com/rowagames/ROWA-Token
Commit	23b3bb5
Whitepaper	https://docs.rowa.games/rowa/
Requirements	https://docs.rowa.games/rowa/
Technical Requirements	https://github.com/rowagames/ROWA-Token/blob/main/contracts/token/Readme.md
Contracts	File: ./contracts/token/RowaToken.sol SHA3: 0088989a7eeb53e7497671915939af01b496b71d1fdad840dd92adee8323211b File: ./contracts/token/RowaVesting.sol SHA3: be4600dcfa4f0fe4adad8d7c24b51e84b60b238180829a2dd3bd65c22ebf7e5c

Third review scope

Repository	https://github.com/rowagames/ROWA-Token
Commit	6e75228
Whitepaper	https://docs.rowa.games/rowa/
Requirements	https://docs.rowa.games/rowa/

Technical Requirements	https://github.com/rowagames/ROWA-Token/blob/main/contracts/token/Readme.md
Contracts	File: ./contracts/token/RowaToken.sol SHA3: f8a9cd14c7c6293a45067028cfd89667dbaefc3b80e03b39059397c6d96c83b1 File: ./contracts/token/RowaVesting.sol SHA3: 526fd16b6804f7f0ffb90f495c94ff8dc1dc97f38d46f4eebd97997ac046140f