HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: RaceKingdom
Date: Jun 24th, 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 Race Kingdom
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC20 token; Vesting
Platform	EVM
Language	Solidity
Methods	Manual Review, Automated Review, Architecture review
Website	https://racekingdom.io
Timeline	16.06.2022 - 24.06.2022
Changelog	16.06.2022 - Initial Review 24.06.2022 - Initial 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 Race Kingdom (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://github.com/racekingdom/smart-contract
Commit:
93057ad60b2938b3f7cdac00bdff7cf907902a16
Technical Documentation:
Type: Token analysis sheet
Link: https://docs.google.com/spreadsheets/d/1KcPYFvoq81n7aUDHzS-fNlwM6N1tf0H4yOEDnYqa-Zk/edit#gid=0

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/RaceKingdom.sol
SHA3: a200b73ac0b013536c70c9eb14fc2512dc1bc2f091c1101e3851c1b281009bbf

File: ./contracts/RKVesting.sol
SHA3: 4bdbb32d9423799bcb131abde45409bf8a35742e7d3cbd7525285d792a4a5a9b

Second review scope

Repository:
https://github.com/racekingdom/smart-contract
Commit:
d44e3f1ad5e6237a424fb8c70406a77542015fa8
Technical Documentation:
Type: Token analysis sheet
Link: https://docs.google.com/spreadsheets/d/1KcPYFvoq81n7aUDHzS-fNlwM6N1tf0H4yOEDnYqa-Zk/edit#gid=0

Type: Whitepaper
Link: https://racekingdom.io/whitepaper-30.pdf
Integration and Unit Tests: Yes
Contracts:
File: ./contracts/RaceKingdom.sol
SHA3: 22a527c20e6ee9815480a2c4a2e435ed7427f8714018cf54708ff99f2f932787

File: ./contracts/RKVesting.sol
SHA3: 54c25ba5023d74d7afeb3108e08fc91002419e3a2547be361dc89eb642d29d90

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

Executive Summary

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

Documentation quality

The Customer provided documentation, that described vesting logic. The total Documentation Quality score is **10** out of **10**.

Code quality

The total CodeQuality score is **10** out of **10**. Code is well-formatted, easy-readable, and without duplications. Good unit test coverage.

Architecture quality

The architecture quality score is **10** out of **10**. Logic is separated by different files, following the single responsibility principle.

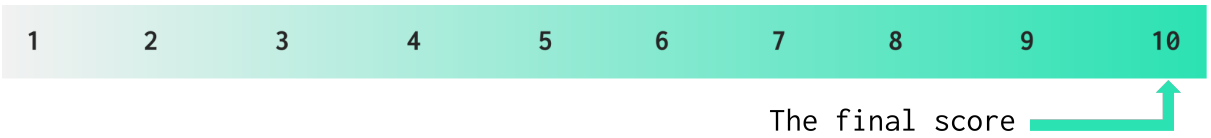
Security score

As a result of the audit, the code contains no 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: **10.0**.



Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are 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.	Passed
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.	Passed
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.	Passed
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.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization	SWC-115	tx.origin should not be used for	Passed

through tx.origin		authorization.	
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 identifier should always be used.	Passed
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.	Passed
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.	Passed
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.	Passed
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	Passed

		fails due to the block Gas limit.	
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
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

Race Kingdom is an animal racing metaverse with the following contracts:

- *RaceKingdom* — simple ERC20 token that mints all initial supply to addresses that will be used in vestings. Additional minting is not allowed.
  It has the following attributes:
  - Name: Race Kingdom
  - Symbol: ATOZ
  - Decimals: 18
  - Total supply: 3.7b tokens.
- RKVesting - a contract that is responsible for the control of vesting periods and unlocks.

Privileged roles

- The owner of the RKVesting contract **can revoke** the vesting schedule for a specific identifier.
- The owner of the RKVesting contract can withdraw exceeded tokens from the vesting pool.
- The owner of the RKVesting contract can create a vesting schedule for a beneficiary.

Risks

- In case of an admin keys leak, an attacker can lock all token transactions or change vestings.

Findings

Critical

No critical severity issues were found.

High

1. **Vesting requirements compliance.**

   As per provided documentation - seed round allocation should be 296m of tokens, but in the *RaceKingdom* contract, it is declared 269m of tokens for the seed round.

   Contracts: RaceKingdom.sol

   Recommendation: Update documentation or seed round allocation.

   Status: Fixed (d44e3f1ad5e6237a424fb8c70406a77542015fa8)

Medium

1. **Stucked funds in the contract.**

   The contract contains payable functions to receive native tokens, but there are no methods to withdraw them from the contract. As a result - all sent native tokens to the contract would be stuck.

   Contracts: RKVesting.sol

   Recommendation: Remove *receive* and *fallback* functions to forbid accidental native coin transfers.

   Status: Fixed (d44e3f1ad5e6237a424fb8c70406a77542015fa8)

Low

1. **The public function could be declared external.**

   Public functions that are never called by the contract should be declared external to save Gas.

   Contracts: RaceKingdom.sol, RKVesting.sol

   Functions: name, symbol, decimals, revoke, withdraw, computeReleasableAmount, getWithdrawableAmount, computeNextVestingScheduleIdForHolder, getLastVestingScheduleForHolder

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

   Status: Fixed (d44e3f1ad5e6237a424fb8c70406a77542015fa8)

2. **Zero address is allowed.**

   The new address for the service signer does not check if it is a zero address, which could be sent as a default value.

Contracts: RKVesting.sol

Functions: createVestingSchedule

Recommendation: Add check for zero address for *_beneficiary.*

Status: Fixed (d44e3f1ad5e6237a424fb8c70406a77542015fa8)

3. **Redundant payable address cast.**

   Release function casts beneficiary address to payable, which is redundant, as contract transfer ERC20 token, not the network native token.

   Contracts: RKVesting.sol

   Functions: release

   Recommendation: Remove *payable* cast before the transfer.

   Status: Fixed (d44e3f1ad5e6237a424fb8c70406a77542015fa8)

4. **Missing vesting validation.**

   createVestingSchedule function does not validate if the cliff period is less than the vesting duration. If the cliff is bigger than the duration - nothing would be released to the beneficiary before the cliff is ended.

   Recommendation: Validate cliff duration when creating a vesting schedule.

   Status: Fixed (d44e3f1ad5e6237a424fb8c70406a77542015fa8)

5. **Variable Shadowing.**

   Solidity allows for ambiguous naming of state variables when inheritance is used. Contract A with a variable x could inherit contract B, which has a state variable x defined. This would result in two separate versions of x, accessed from contract A and the other from contract B. In more complex contract systems, this condition could go unnoticed and subsequently lead to security issues.

   Contracts: RaceKingdom.sol

   Functions: allowance(address **owner**) -> Ownable.**owner**(),
   approve(address **owner**) -> Ownable.**owner**(),
   transfer(address **owner**) -> Ownable.**owner**(),
   increaseAllowance(address **owner**) -> Ownable.**owner**(),
   decreaseAllowance(address **owner**) -> Ownable.**owner**(),
   _approve(address **owner**) -> Ownable.**owner**(),
   _spendAllowance(address **owner**) -> Ownable.**owner**(),

   Recommendation: Consider renaming the function argument.

   Status: Fixed (d44e3f1ad5e6237a424fb8c70406a77542015fa8)

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.