SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Kingdom Raids
Date: December 15th, 2021

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 Kingdom Raids.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC20 token; ERC721 token; Vesting
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/kingdomraids/kr-nft https://github.com/kingdomraids/kr-token https://github.com/kingdomraids/kr-ido-contract
Commit	6dafe413dd90e5b1c1e85a5b8ec6c8fc71fd87af a2c25ce00ca5a02ae36e1275243e83a661fad6d9 b8e0b81319209fc0e3144d336d3a9e0bcc808b62
Technical Documentation	NO
JS tests	NO
Website	Kingdomraids.io
Timeline	30 NOVEMBER 2021 - 15 DECEMBER 2021
Changelog	06 DECEMBER 2021 - Initial Audit 15 DECEMBER 2021 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Kingdom Raids (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 contract and its code review conducted between November 30th, 2021 - December 06th, 2021.

Second review conducted on December 15th, 2021.

Scope

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

Repository:
https://github.com/kingdomraids/kr-nft
https://github.com/kingdomraids/kr-token
https://github.com/kingdomraids/kr-ido-contract
Commit:
6dafe413dd90e5b1c1e85a5b8ec6c8fc71fd87af
a2c25ce00ca5a02ae36e1275243e83a661fad6d9
b8e0b81319209fc0e3144d336d3a9e0bcc808b62
Technical Documentation: No
JS tests: No
Contracts:
Hero/Hero.sol
Interfaces/IHero.sol
Summon.sol
KRToken.sol
Metric/EcosystemFund.sol
Metric/Team.sol
Metric/Advisor.sol
Metric/Liquidity.sol
Metric/Marketing.sol
Metric/PrivateSale.sol
Metric/CompanyReserves.sol
Metric/SeedSale.sol
IDO.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 ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are well-secured.

Insecure	Poor secured	Secured	Well-secured

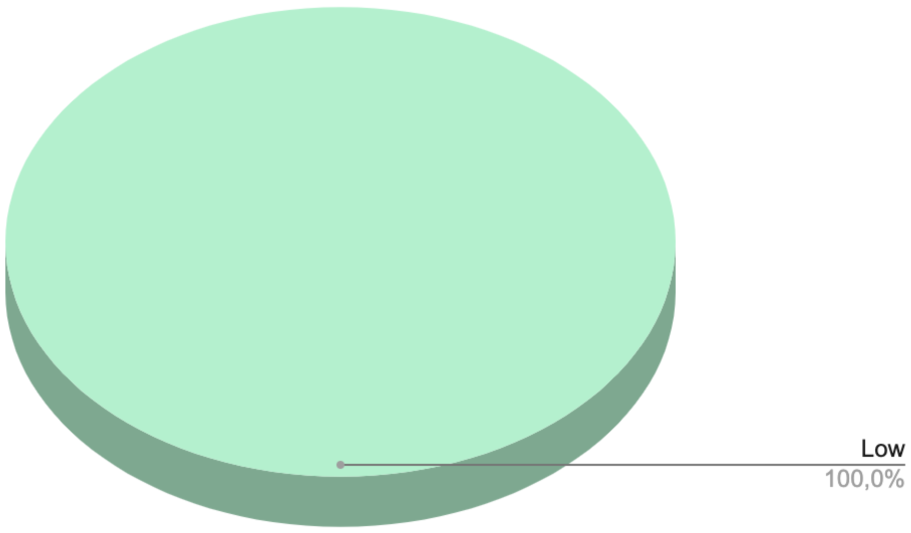
You are here ↑

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

As a result of the audit, security engineers found **1** medium and **10** low severity issues.

After the second review, security engineers found **2** low severity issues.

Graph 1. The distribution of vulnerabilities after the audit.



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

Audit overview

■ ■ ■ ■ Critical

No critical issues were found.

■ ■ ■ High

No high severity issues were found.

■ ■ Medium

1. Lock contracts don’t enforce allocation schedules.

   Contracts: EcosystemFund.sol, Liquidity.sol, Marketing.sol, PrivateSale.sol, SeedSale.sol, Advisor.sol, Team.sol

   Functions: unlock

   The owner could call *unlock* after *release* to change *nextTimeRelease* and be able to call *release* again etc., up to withdraw the entire balance.

   Recommendation: Disallow calling *unlock* more than once.

   Status: fixed

■ Low

1. Syntax error

   Contracts: CompanyReserves.sol

   Functions: release (lines #38,#39)

   Recommendation: fix variable name

   Status: fixed

2. Misleading revert message

   Contracts: IDO.sol

   Functions: constructor (lines #81,#82)

   Require statement check *_startRedeemAt* **<** *_endRedeemAt*, but revert message states *_startRedeemAt must be* **<=** *_endRedeemAt*

   Recommendation: change revert message

3. Using SafeMath in Solidity >= 0.8.0

   Starting solidity version 0.8.0 arithmetic operations revert on underflow and overflow. There's no more need to assert the result of operations.

   Contracts: Hero.sol, EcosystemFund.sol, Liquidity.sol, Marketing.sol, PrivateSale.sol, SeedSale.sol, Advisor.sol, Team.sol, Summon.sol

   Recommendation: Please avoid using assert for arithmetic operations.

   Status: fixed

4. Misleading comment

   Contracts: IDO.sol (lines #26,#27)

   Comment before variable belongs to a different variable.

   Recommendation: change comment

   Status: fixed

5. State variables that could be declared constant

   Constant state variables should be declared constant to save gas.

   Contracts: EcosystemFund.sol, Liquidity.sol, Marketing.sol, PrivateSale.sol, SeedSale.sol, Advisor.sol, Team.sol

   Variables: eachReleaseAmount, releasePeriod

   Recommendation: Add the constant attributes to state variables that never change.

   Status: fixed

6. Boolean equality

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

   Contracts: IDO.sol, CompanyReserves.sol

   Functions: redeemable, release

   Recommendation: remove the equality to the boolean constant.

   Status: fixed

7. A public function that could be declared external.

   public functions that are never called by the contract should be declared **external** to save gas.

   Contracts: Summon.sol, Hero.sol

Functions: setSignerPublicKey, setHeroSmartContractAddress, setFee, setSupplyLimit

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

8. Missing event for changing *_supplyLimit*, *signerPublicKey*, *acceptedToken*, *heroSmartContractAddress*, *fee*

   Contracts: Hero.sol, Summon.sol

   Functions: setSupplyLimit, setSignerPublicKey, setAcceptedToken, setHeroSmartContractAddress, setFee

   Changing critical values should be followed by the event emitting for better tracking off-chain.

   Recommendation: Please emit events on the critical values changing.

   Status: fixed

9. View function returns an array of unpredictable size

   Contracts: Hero.sol

   Functions: walletOfOwner

   Starting from a certain amount of tokens owned by a single user function could become inoperable.

   Recommendation: Add *limit* and *offset* parameter to view function

   Status: fixed

10. Missing validation

    Contracts: Summon.sol

    Functions: setHeroSmartContractAddress

    Address validated during contract creation but not in setter method

    Recommendation: add *isContract()* validation

    Status: fixed

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

As a result of the audit, security engineers found **1** medium and **10** low severity issues.

After the second review, security engineers found **2** low severity issues.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to 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 as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. 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 can't guarantee the explicit security of the audited smart contracts.