HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Digital Arms
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 Digital Arms.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC20 token; Vesting
Platform	Binance / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/grape404/Hunters-Pre-Sale-Vesting
Commit	5498d05494fecf4369b11262fce93f73e9f517c6
Technical Documentation	YES
JS tests	YES
Website	Hunter-token.com
Timeline	15 NOVEMBER 2021 - 15 DECEMBER 2021
Changelog	18 NOVEMBER 2021 - Initial Audit 26 NOVEMBER 2021 - Second Review 15 DECEMBER 2021 - Third 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 Digital Arms (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 15th, 2021 - November 18th, 2021.

Second review conducted on November 26th, 2021.

Third review conducted on December 15th, 2021.

Scope

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

Repository:
https://github.com/grape404/Hunters-Pre-Sale-Vesting
Commit:
5498d05494fecf4369b11262fce93f73e9f517c6
Technical Documentation: Yes, https://magnetic-sea-006.notion.site/Hunters-Pre-Sale-Vesting-Smart-Contracts-Specification-and-Functions-Document-c250318ebc5d4a7e929d0c9f43d334c1
https://docsend.com/view/gcfdaiymvyqcakkh
JS tests: Yes, in the repository
Contracts:
Vesting.sol
ERC20Token.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

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

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.

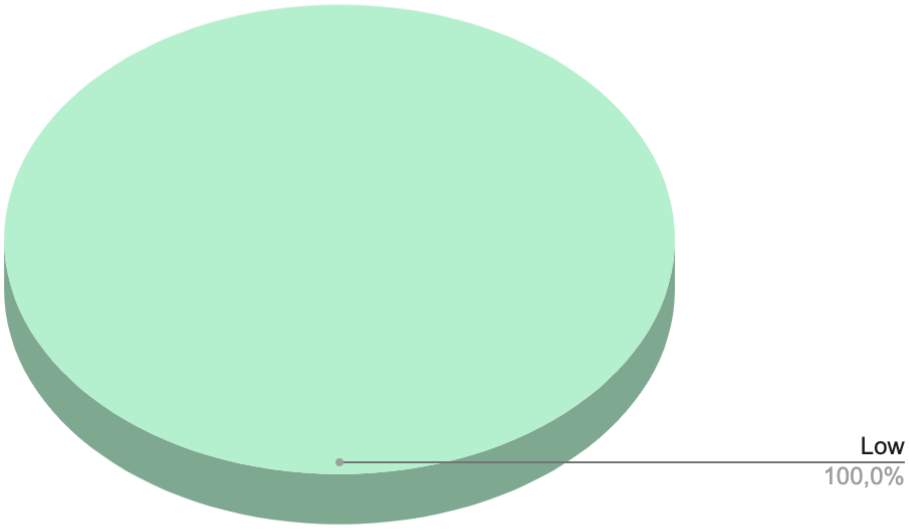
As a result of the audit, security engineers found **1** medium and **6** low severity issues.

As a result of the second review, security engineers found **1** low severity issue.

As a result of the third review, security engineers found that functionality was slightly changed. Therefore found **2** low severity issues.

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

Graph 1. The distribution of vulnerabilities after the audit.



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 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

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

Audit overview

Critical

No critical issues were found.

High

No high severity issues were found.

Medium

In some conditions, users could bypass the vesting monthly allocation mechanism.

After the last call of *setDistributionPercent*, when *tierVestingInfo[_tierId].totalAllocationDone* equals 10000, but before *setVestingTimeForTier* called by owner, any user that already bought vesting tokens could call *vestTokens* to receive all tokens regardless of schedule.

Contracts:Vesting.sol

Functions: vestTokens

Recommendation: add a check **tierVestingInfo[_tierId].vestingStartTime != 0**

Status: fixed

Low

1. Possible token loss

In case *tierVestingInfo[_tierId].vestingStartTime* is earlier than *tierInfo[_tierId].endTime* for some *_tierId,* between these moments user could call *buyVestingTokens* after *vestTokens* which results loss of some fraction of his/her tokens.

Contracts:Vesting.sol

Recommendation: add a check that prevents such a scenario.

Status: fixed

2. Possible data inconsistency

If function *setDistributionPercent* calls more than once for same month, sum of *allocationPerMonth[_tierId][]* become not equal *tierVestingInfo[_tierId].totalAllocationDone*

Contracts:Vesting.sol

Recommendation: add a check that prevents the second call for the same month or include the current value in the calculation

Status: fixed

3. Misleading revert message

Contracts: Vesting.sol

Functions: vestTokens (line #390)

Recommendation: change value to **10000** or **100%**

Status: fixed

4. State variables that could be declared constant

Constant state variables should be declared constant to save gas.

Contracts: Vesting.sol

Variables: secondsInMonth

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

Status: fixed

5. 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: Vesting.sol

Functions: whitelistAddress, removeWhitelistAddress, setDistributionPercent, setVestingTimeForTier, buyVestingTokens, vestTokens, adminWithdrawStableCoin

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

Status: fixed

6. 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: Vesting.sol

Recommendation: Please avoid using assert for arithmetic operations.

7. Boolean equality

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

Contracts: Vesting.sol

Functions: buyVestingTokens, allocateVestingTokens, removeAllocatedVestingTokens

Recommendation: remove the equality to the boolean constant.

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

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 **6** low severity issues.

As a result of the second review, security engineers found **1** low severity issue.

As a result of the third review, security engineers found that functionality was slightly changed. Therefore found **2** low severity issues.

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 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.