HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: DotFinance
Date: September 28th, 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 DotFinance.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC20 token; Transfer controller; DEFI
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/Dot-Finance/Dot
Commit	8FB1031EB9D874233B75EFC744279A679747D54B
Technical Documentation	YES
JS tests	NO
Timeline	24 AUG 2021 - 28 SEP 2021
Changelog	13 SEP 2021 - INITIAL AUDIT 28 SEP 2021 - 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 DotFinance (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 August 26th, 2021 - September 13th, 2021. The second code review conducted on September 28th, 2021.

Scope

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

Repository:
https://github.com/Dot-Finance/Dot
Commit:
8fb1031eb9d874233b75efc744279a679747d54b
Technical Documentation: Yes
JS tests: No

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 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 **2** high, **2** medium, **2** low, and **4** informational severity issues.

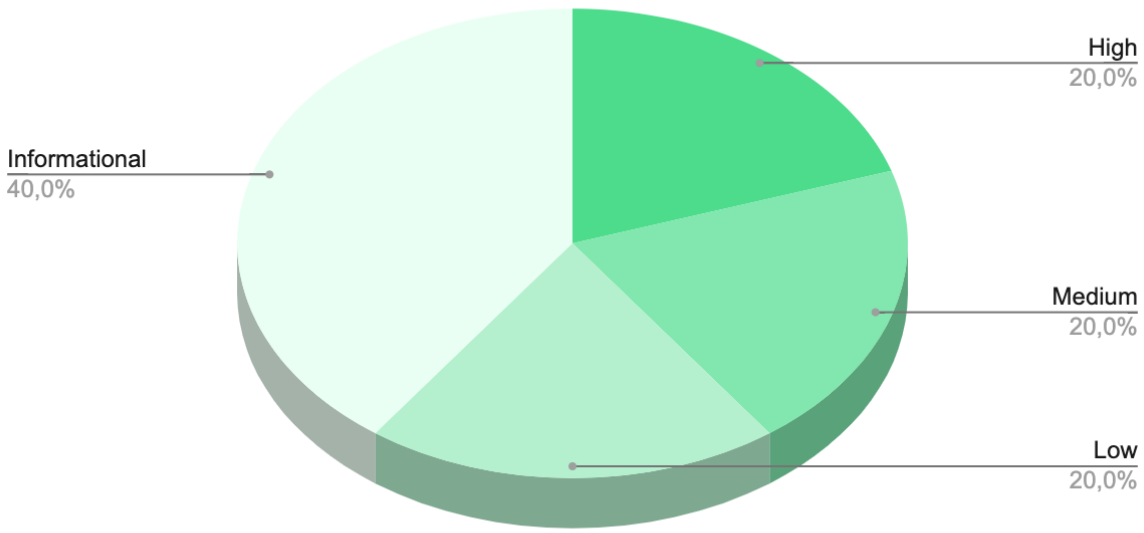
As a result of the second review, security engineers found **2** medium, **1** low, and **4** informational severity issues.

Notice:

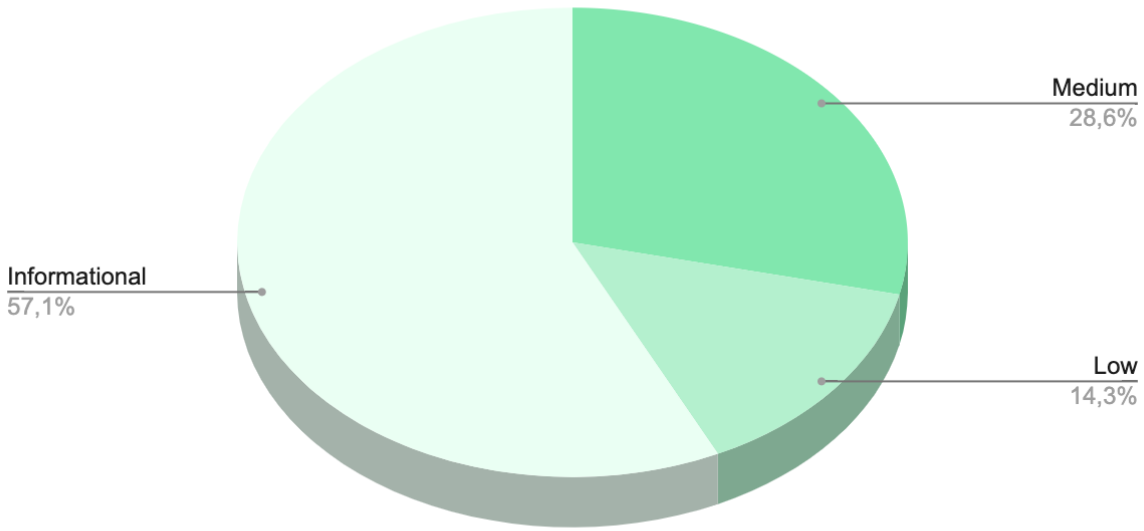
The project contains neither Unit Tests nor other types of tests. We strongly recommend the Customer cover at least the main functionality with unit tests.

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

Graph 1. The distribution of vulnerabilities after the audit.



Graph 2. The distribution of vulnerabilities after the second review.



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

1. The compiler version should be updated to the latest.

   Status: fixed

2. There is the ability to change the Helper version of PinkPool.sol after the contract was deployed and used by users.

   Contracts:PinkPool.sol

   Function: setHelper

   Status: fixed.

Medium

1. Reward token decimals value is hardcoded in function apy() of PinkPool.sol contract, but reward token can be changed and set by the owner.

   Contracts:PinkPool.sol

   Function: apy

   Status: open.

2. There are lots of hardcoded ‘magic values’ like 1e18 in the contracts. The hardcoded values should be removed from functions especially if they are related to changeable entities.

   Status: open.

Low

There are set of values like:
   a. pinkPrice
   b. flipPrice
   c. rewardPerToken

which are calculated or received from helper (by calling another contract). These values can be cached (saved to fields) by block time to decrease gas usage.

Contracts:PinkPool.sol

Status: open.

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

Informational

1. We strongly recommend you add proper values of error messages in require validation (related to all code).
2. It is better to use language construction for the operations it was created for. Modifiers provide the way to do some validations and restrictions, due to solidity language philosophy. We recommend you change the modifiers like PinkPool.updateReward to function view.
3. It may be better to check the balance and validate the existence of the needed amount of pink token, before calling _flipToPink in the getReward function of PinkPool.sol contract.
4. Event emission should be added to all functions which change the pool configurations.

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.

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

As a result of the audit, security engineers found **2** high, **2** medium, **2** low, and **4** informational severity issues.

As a result of the second review, security engineers found **2** medium, **1** low, and **4** informational severity issues.

Notice:

The project contains neither Unit Tests nor other types of tests. We strongly recommend the Customer cover at least the main functionality with unit tests.

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, bugfree 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.