SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Nimbus
Date: July 16th, 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 Nimbus.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	dApps
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/nimbusplatformorg/nim-smartcontract/tree/7bda71190cca5d139e15b46a33ca041eb060f38d (Initial Audit) https://github.com/nimbusplatformorg/nim-smartcontract/commit/12a41a194f39670637d79ef2c5bcc6a70d617781 (Second review)
Deployed contract	
Changelog	24 May 2021 – Initial Audit 16 July 2021 – Second Review

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 Nimbus (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer's smart contract and its code review conducted on May 24th, 2021.

Second review conducted on July 16th, 2021.

Scope

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

```
Repository: https://github.com/nimbusplatformorg/nim-smartcontract/commit/
Commit: 12a41a194f39670637d79ef2c5bcc6a70d617781

Files:
      dApps/P2P/NimbusP2PERC20.sol
      dApps/RevenueChannels/*
```

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 secure.

Insecure	Poor secured	Secured	Well-secured

You are here ↑ Secured

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 high, 2 medium and 6 low severity issues.

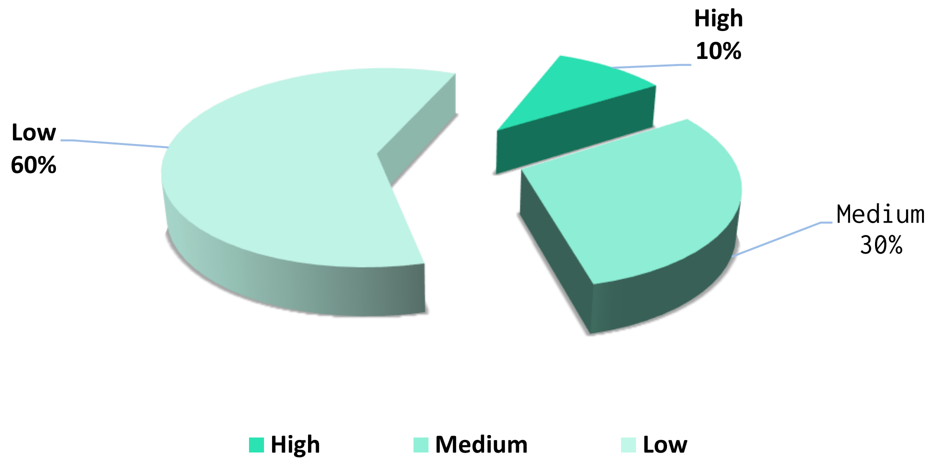
After the second review the code contains 2 medium and 6 low severity issues.

Notices:

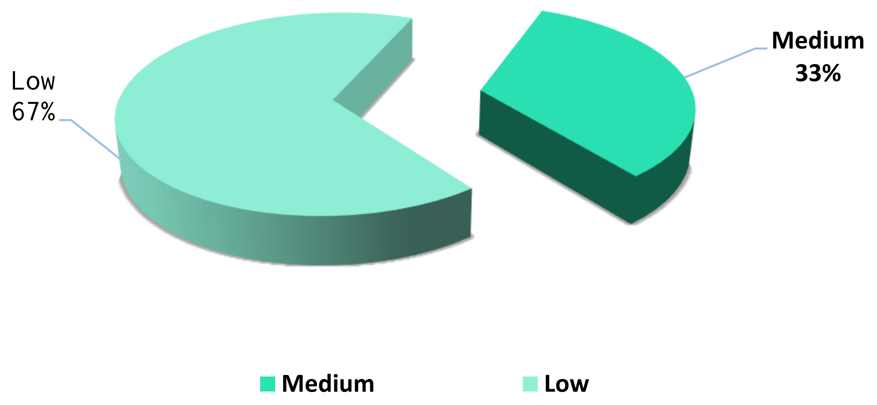
1. Description of contracts logic is not provided by the Customer and we may not prove correctness of some calculation.

2. The code is not covered with unit tests. We strongly recommend covering as much code as possible.

Graph 1. The distribution of vulnerabilities after the audit.



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



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

1. swapsImpl contract is not in the audit scope. Results of the dexSwap function call is used in unsafe math operations in the _swapsCall function. Uint256 overflow can happen.

   Contracts: SwapsUsers.sol

   Function: _swapsCall_internal

   Recommendation: Provide the SwapsImpl contract code or use safe math operations.

   Status: Addressed in 12A41A194F39670637D79EF2C5BCC6A70D617781 commit.

🟩🟩 Medium

1. Hardcoded addresses are used.

   Contracts: LoanTokenLogicStandard.sol, LoanTokenSettingsLowerAdmin.sol

   Recommendation: Init values in constructor or in a separate init function.

2. incentivePercent value is not validated for 0. As soon as its value in the liquidationIncentivePercent is set by owners manually, there's no guarantee that a value exists when the function is called.

   Contracts: LiquidationHelper.sol

   Function: _getLiquidationAmounts

   Recommendation: add non-zero validation

🟦 Low

1. Contracts uses old compiler version.

   Contracts: all

   Recommendation: update to the latest stable compiler version.

2. The contract contains no public or external functions.

Contracts: SwapsUser, VaultController, LiquidationHelper, InterestUser, FeesHelper, EnumerableBytes32Set

Recommendation: mark contracts as abstract.

3. Copies of the OpenZeppelin contracts are stored.

   Recommendation: import all libraries directly from the OpenZeppelin. Get rid of local copies.

4. Contracts are unused.

   Contracts: IChai.sol

   Recommendation: remove unused contracts.

5. Errors with number like “15” and “16” are not descriptive.

   Contracts: all

   Recommendation: throw descriptive error messages.

6. The contract should be an interface. The LoanMaintenance should implement this interface.

   Contracts: ProtocolLike.sol

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 1 high, 2 medium and 6 low severity issues.

After the second review the code contains 2 medium and 6 low severity issues.

Notices:

1. Description of contracts logic is not provided by the Customer and we may not prove correctness of some calculation.
2. The code is not covered with unit tests. We strongly recommend covering as much code as possible.

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 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 security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.