SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: NasDex
Date: October 4th, 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 NasDex.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	MasterChef
Platform	Polygon / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/NasDex/NasDex
Commit	b575cab1663d4f5689c4650c42abb0d67adf879d
Technical Documentation	NO
JS tests	YES
Timeline	27 SEPTEMBER 2021 - 04 OCTOBER 2021
Changelog	01 OCTOBER 2021 - Initial Audit 04 OCTOBER 2021 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by NasDex (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 September 27th, 2021 - October 1st, 2021.

Second review conducted on October 4th, 2021.

Scope

The scope of the project is smart contracts in the repository:
Repository:
    https://github.com/NasDex/NasDex
Commit:
    b575cab1663d4f5689c4650c42abb0d67adf879d
Technical Documentation: No
JS tests: Yes
Contracts:
    MasterChef.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.

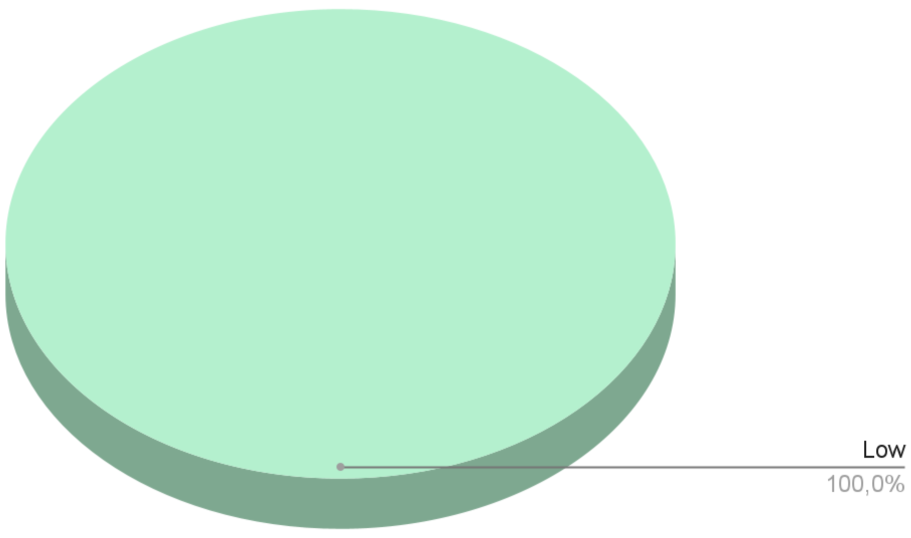
Insecure	Poor secured	Secured	Well-secured
		You are here	↑

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** high and **3** 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

Possible rewards lost or receive more

Changing **allocPoint** in the MasterChef.set method while **_withUpdate** flag set to **false** may lead to rewards lost or receiving rewards more than deserved.

Recommendation: Please call updatePool(_pid) in the case if **_withUpdate** flag is **false** and you don't want to update all pools.

Fixed before the second review.

■ ■ Medium

No medium severity issues were found.

■ Low

1. Unnecessary operations

   When allocPoint is not changed for the pool, there still sub and add functions called on the totalAllocPoint, which just consumes gas doing nothing.

   Recommendation: Please move totalAllocPoint calculation to the *if (poolInfo[_pid].allocPoint != _allocPoint)* block.

   Fixed before the second review.

2. Missing events on important values changed in contracts

   Changing important values, such as BONUS_MULTIPLIER, startTimestamp, governance should emit events that will allow for the community to track off-chain and react to the changes.

   Recommendation: Please emit events on changes.

   Fixed before the second review.

3. State variables that could be declared immutable

   State variables that never change their values and are initialized in the constructor should be declared **immutable** to save gas.

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

Lines: #47

```
uint256 public nsdxPerBlock;
```

4. Implicit visibility

   State variables that don't have visibility specified in the declaration are implicitly set visibility as internal. While before the second review the visibility of the variables below was changed from the **public** to **internal** which leads to that those variables could not be viewed by the public.

   Recommendation: Please add an explicit visibility declaration to avoid confusion.

Lines: #40-44

```
// The NSDX TOKEN!
NSDXToken immutable nsdx;

// The reward bar
NSDXBar immutable bar;
```

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** high and **3** 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.