Review Report

04/02/2025

Flamingo Flocks Review 04/02/2025

Content

Introduction

Flamingo is a platform that helps convert tokens, be a liquidity provider and earn yield. By providing liquidity, also known as staking, you earn yield by collecting fees and getting minted FLM as a reward. Flamingo Finance makes it easy to buy/sell crypto, invest and earn revenue directly on the blockchain.

Flocks is a dividend-bearing asset, it is defined as grabbing a share of **Flamingo**. The objective is that a **Flamingo** user can buy a share of **Flocks** to receive passive income. The intention is that the yield comes as a **Flocks** holder from different sources: platform fees, interests, and minting rewards.

As solicited by **Neo** and as part of the vulnerability review and management process, Red4Sec has been requested to perform a security assessment in order to evaluate the security of the **Flamingo Flocks** project.

The report includes specifics retrieved from the review for all the existing vulnerabilities of **Flamingo Flocks**. The performed analysis shows that the smart contract does contain high risk vulnerabilities.

Disclaimer

This document only represents the results of the security assessment conducted by Red4Sec Cybersecurity and should not be used in any way to make investment decisions or as investment advice on a project.

Likewise, the report should not be considered either "endorsement" or "disapproval" of the guarantee of the correct business model of the analyzed project, nor as guarantee on the operation or viability of the implemented financial product.

Red4Sec makes full effort and applies every resource available for each audit and review, however it does not warrant the function, nor the safety of the project and it cannot be deemed a sufficient assessment of the code's utility and safety, bug-free status, or any other declarations of the project. Additionally, Red4Sec makes no security assessments or judgments about the underlying business strategy, or the individuals involved in the project.

Blockchain technology and cryptographic assets come with their own new risks and challenges, where the ecosystem, platform, its programming language, and other software related to said technology can have vulnerabilities that could lead to exploits. As a result, the audit cannot guarantee the explicit security of the audited projects.

The review reports can be used to improve the code quality of smart contracts, to help limit the vectors of attack and to lower the high level of risks associated with utilizing new and continually changing technologies such as cryptographic tokens and blockchain, but they are unable to detect any future security concerns with the related technologies.

Scope

Red4Sec Cybersecurity has made a security assessment of the **Flamingo Flocks** security level against attacks, identifying possible errors in the design, configuration, or programming; therefore, guaranteeing the availability, integrity and confidentiality of the project and the possible assets treated and stored.

The scope of this evaluation includes the following items provided by **Neo**:

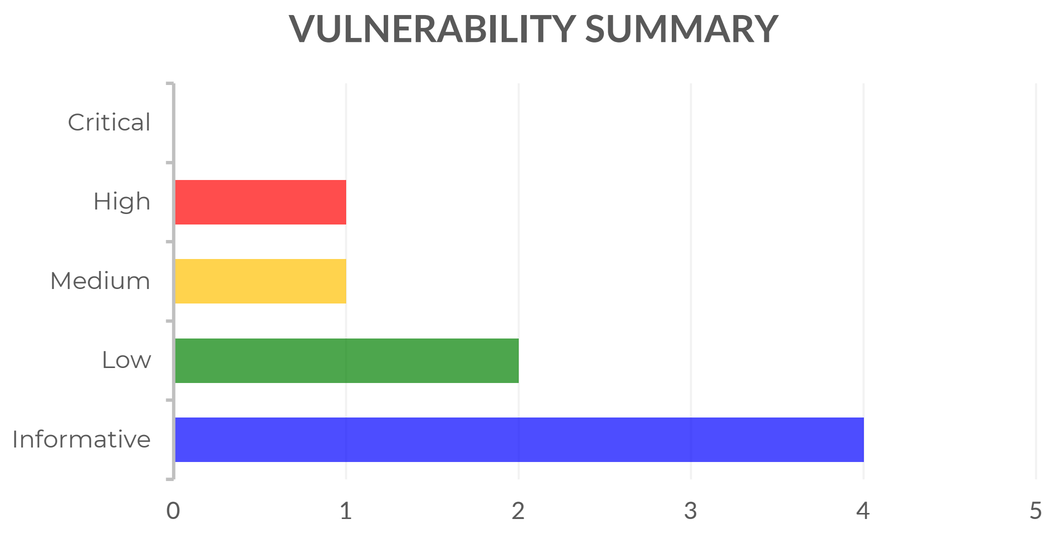
- **Flocks Smart Contract**
  - Code: https://github.com/flamingo-finance/flamingo-sc-monorepo/tree/features/flocks
  - Commit: `b168f125509815884574b89600e959697a97459f`
  - Fixes Review: `696849002758c9ca40a607b923616853d286ad6e`

Executive Summary

The security assessment against **Flamingo Flocks** has been conducted between the following dates: **20/01/2025** and **04/02/2025**.

Once the analysis of the technical aspects has been completed, the performed analysis shows that the reviewed source code contains high risk vulnerabilities that should be mitigated as soon as possible.

During the analysis, a total of **8 vulnerabilities** were detected, these vulnerabilities have been classified by the following level of risks, defined in Vulnerabilities Severity annex.



Vulnerabilities

In this section, you can find a detailed analysis of the vulnerabilities encountered upon the security assessment.

List of vulnerabilities

Below, we have gathered a complete list of the vulnerabilities detected by Red4Sec, presented, and summarized in a way that can be used for risk management and mitigation.

Table of vulnerabilities			
ID	**Vulnerability**	**Risk**	**State**
FLOCKS-01	Reentrancy by NoReentrant Attribute	**High**	**Fixed**
FLOCKS-02	Key Variable on Storage	**Medium**	**Fixed**
FLOCKS-03	Denial of Service in the Query Methods	**Low**	**Assumed**
FLOCKS-04	Testnet Private Key Leakage	**Low**	**Assumed**
FLOCKS-05	Lack of Inputs Validation	**Informative**	**Open**
FLOCKS-06	Using _deploy Method for Initial Configuration	**Informative**	**Open**
FLOCKS-07	Bad Coding Practices	**Informative**	**Partially Fixed**
FLOCKS-08	Missing Manifest Information	**Informative**	**Assumed**

Vulnerability details

In this section, we provide the details of each of the detected vulnerabilities indicating the following aspects:

- Category
- Active
- Risk
- Description
- Recommendations

Reentrancy by NoReentrant Attribute

Identifier	Category	Risk	State
FLOCKS-01	Denial of Service	**High**	**Fixed**

In versions higher than **3.7.4**, the `NoReentrant` attribute is global, meaning it cannot be called from a method that already has this attribute. Doing so would trigger a Denial of Service, as it would be detected as a re-entry to said method.

It is recommended to use an internal method without these protections if the call to the same logic is required, in this way the method can be called from the contract itself without reentry protections, while keeping the public method protected.

References

- devpack-dotnet/issues/1182
- devpack-dotnet/pull/1181
- NoReentrantAttribute.cs
- NoReentrantMethodAttribute.cs

Source Code References

- Flamingo.FLOCKS/FLOCKS.Burn.cs#L23
- Flamingo.FLOCKS/FLOCKS.Account.cs#L45
- Flamingo.FLOCKS/FLOCKS.Account.cs#L69
- Flamingo.FLOCKS/FLOCKS.Claim.cs#L22
- Flamingo.FLOCKS/FLOCKS.Mint.cs#L58
- Flamingo.FLOCKS/FLOCKS.Mint.cs#L90
- Flamingo.FLOCKS/FLOCKS.Mint.cs#L133
- Flamingo.FLOCKS/FLOCKS.Nep17Token.cs#L51

Key Variable on Storage

Identifier	Category	Risk	State
FLOCKS-02	Codebase Quality	**Medium**	**Fixed**

Using variable-length variables to generate storage keys is not recommended, as it can lead to key storage collisions.

It has been detected that a `BigInteger` cast to `ByteString` is concatenated with a `UInt160`, the underlying problem here is that *[0,0,1]* or *[1]* as a `ByteString` represent the same valid `BigInteger`, but result in two entirely different storage keys.

The use of variable length variables to generate storage keys can cause storage key collisions. This happens because the generated keys may not be unique or predictable, which can lead to unintentional data overwrites or unexpected access to sensitive information. It is recommended to use fixed identifiers or deterministic hashes of unique values to ensure storage integrity on the blockchain.

Recommendations

It is convenient to use the practice of unifying all the prefixes to the same type and of the same length to avoid possible collisions and injections in the use of storage keys.

Source Code References

- Flamingo.FLOCKS/FLOCKS.Storage.cs#L253-L259
- Flamingo.FLOCKS/FLOCKS.Storage.cs#L271-L277
- Flamingo.FLOCKS/FLOCKS.Storage.cs#L364-L371
- Flamingo.FLOCKS/FLOCKS.Storage.cs#L396-L402
- Flamingo.FLOCKS/FLOCKS.Storage.cs#L688-L694

Denial of Service in the Query Methods

Identifier	Category	Risk	State
FLOCKS-03	Denial of Service	**Low**	**Assumed**

Different query methods of the contract return a list of characteristics without considering that Neo's virtual machine has a limitation of *2048* elements in the return of the operations and a Denial of Service could occur with higher values.

Taking into consideration a scenario where these query methods fail at some point by exceeding the limits of the virtual machine and returning `FAULT` in its execution, a Denial of Service would be produced, this limits the affected methods to fewer than 600 entries.

Source Code References

- Flamingo.FLOCKS/FLOCKS.Claim.cs#L45
- Flamingo.FLOCKS/FLOCKS.Account.cs#L92
- Flamingo.FLOCKS/FLOCKS.cs#L132

Testnet Private Key Leakage

Identifier	Category	Risk	State
FLOCKS-04	Testing and Documentation	**Low**	**Assumed**

Testnet private key leakage was discovered during the security assessment, indicating that a private key used on the testnet was exposed, potentially compromising test transactions.

Source Code References

- localnet.neo-express.js#L15-L17
- utils/testnet-ctl.js#L10

Lack of Inputs Validation

Identifier	Category	Risk	State
FLOCKS-05	Data Validation	**Informative**	**Open**

Certain methods do not properly check the arguments, which can lead to major errors.

In various methods it is convenient to check that the value is not `IsZero`, leaving the verification as: `hash.IsValid && !hash.IsZero`.

Recommendations

It is advisable to always check the format of the arguments before using their value, otherwise, a user could send unexpected values through these arguments, being able to make injections or arbitrary reads from the storage, either intentionally or not.

Source Code References

- Flamingo.FLOCKS/FLOCKS.Claim.cs#L66
- Flamingo.FLOCKS/FLOCKS.Claim.cs#L83
- Flamingo.FLOCKS/FLOCKS.cs#L108
- Flamingo.FLOCKS/FLOCKS.cs#L146
- Flamingo.FLOCKS/FLOCKS.cs#L179
- Flamingo.FLOCKS/FLOCKS.cs#L193
- Flamingo.FLOCKS/FLOCKS.cs#L211

Using _deploy Method for Initial Configuration

Identifier	Category	Risk	State
FLOCKS-06	Testing and Documentation	**Informative**	**Open**

Modifying contract code directly via an external script using regex, rather than utilizing the `_deploy` method to provide configuration arguments, is considered a bad practice. Additionally, relying on code replacements with tools such as regex can lead to inconsistencies if spaces or formats vary.

It is recommended to set configuration parameters through explicit initialization methods.

Source Code References

- Flamingo.FLOCKS/build.js#L75

Bad Coding Practices

Identifier	Category	Risk	State
FLOCKS-07	Codebase Quality	**Informative**	**Partially Fixed**

During the security assessment of the smart contract, certain bad practices have been detected throughout the code that should be improved. It is strongly advised to consider the adoption of improved coding styles and best practices for overall code improvement.

Check `pause`/`unpause`:

- Flamingo.FLOCKS/FLOCKS.Owner.cs#L122-L205

Use inheritance in storage:

- Flamingo.FLOCKS/FLOCKS.Storage.cs

Unshorted storage indexes:

- Flamingo.FLOCKS/FLOCKS.Storage.cs#L193

Unify Decrease underflow check:

- Flamingo.FLOCKS/FLOCKS.Storage.cs#L465
- Flamingo.FLOCKS/FLOCKS.Storage.cs#L519
- Flamingo.FLOCKS/FLOCKS.Storage.cs#L675

Unify arguments order. `user`, `amount` should be better:

- Flamingo.FLOCKS/FLOCKS.Helpers.cs#L16

Missing Manifest Information

Identifier	Category	Risk	State
FLOCKS-08	Codebase Quality	**Informative**	**Assumed**

The contract does not properly specify the information related to the Smart Contract in its manifest.

All the contract information identified in the contract's manifest (*author, email, description, source*) belongs to the developer of the project. Therefore, it is recommended to customize said content by adding your own information, which will also provide the users with relevant information about the project.

Recommendations

- Adding all possible metadata to the contracts favors indexing, traceability, and the audit by users, which conveys greater confidence to the user.

References

- nep-16.mediawiki#source

Source Code References

- Flamingo.FLOCKS/FLOCKS.cs#L12-L16

RED4SEC

Invest in Security, invest in your future