

SALUS SECURITY

MAY 2023



CODE SECURITY ASSESSMENT

FLAPPYMOONBIRD

Overview

Project Summary

- Name: FlappyMoonBird
- Platform: Ethereum
- Language: Solidity
- Audit Range: See [Appendix - 1](#)

Project Dashboard

Application Summary

Name	FlappyMoonBird
Version	v1
Type	Solidity
Dates	May 29 2023
Logs	May 29 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	1
Total informational issues	1
Total	2

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction	4
1.1 About SALUS	4
1.2 Audit Breakdown	4
1.3 Disclaimer	4
Findings	5
2.1 Summary of Findings	5
2.2 Notable Findings	6
1. Centralization risk with initial token distribution	6
2.3 Informational Findings	7
2. Magic numbers are used	7
Appendix	8
Appendix 1 - Files in Scope	8

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (<https://t.me/salusec>), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Centralization risk with initial token distribution	Low	Centralization	Pending
2	Magic numbers are used	Informational	Code Quality	Pending

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Centralization risk with initial token distribution	
Severity: Low	Category: Centralization
Target: <ul style="list-style-type: none">- FMB.sol	

Description

When the contract is deployed, 41% of \$FMB is sent to the `COMMUNITY_ADDRESS`. This address then has full control over the community rewards distribution, so any compromise of its private key could have a drastic effect on the project – for example, attackers could dump the price of \$FMB on the DEX if they gain access to the private key.

The same centralization concern applies to `ECOSYSTEM_ADDRESS` and `MARKETING_ADDRESS` too.

Recommendation

Consider setting these addresses to multi-sig accounts so that the initial tokens, which will be redistributed later, are sent to multi-sig accounts, rather than EOA accounts.

2.3 Informational Findings

2. Magic numbers are used

Severity: Informational

Category: Code Quality

Target:

- FMB.sol

Description

There are a few occurrences of literal values being used. To improve the code's readability and facilitate refactoring, consider defining a constant for every magic number, giving it a clear and self-explanatory name.

FMB.sol:L27-L34

```
constructor() ERC20("Flappy Moon Bird", "FMB") {
    _mint(PRE_SEED_A_ADDRESS, TOTAL_SUPPLY * PRE_SEED_A_PERCENT / 100);
    _mint(PRE_SEED_B_ADDRESS, TOTAL_SUPPLY * PRE_SEED_B_PERCENT / 100);
    _mint(COMMUNITY_ADDRESS, TOTAL_SUPPLY * COMMUNITY_PERCENT / 100);
    _mint(DEVELOPMENT_ADDRESS, TOTAL_SUPPLY * DEVELOPMENT_PERCENT / 100);
    _mint(ECOSYSTEM_ADDRESS, TOTAL_SUPPLY * ECOSYSTEM_PERCENT / 100);
    _mint(MARKETING_ADDRESS, TOTAL_SUPPLY * MARKETING_PERCENT / 100);
}
```

Recommendation

Consider defining a constant variable FULL_PERCENTAGE for the magic number 100.

Appendix

Appendix 1 - Files in Scope

This audit covered the following file:

File	SHA-1 hash
FMB.sol	10c1d2d621688509053651471f5dd89c9d300a61