# Audit Whale

Smart contract code review and security analysis report

## Document

| Name | Smart Contract Code Review and Security Analysis Report for Fancy. |
|---|---|
| Approved by | Jameson \| COO AuditWhale |
| Type | ERC20 token; Staking |
| Platform | Ethereum / Solidity |
| Methods | Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review |
| Repository | https://github.com/Fancy-Birds/staking |
| Commit | ea37a4b54e8fd81559361948ca7ed2e556a29e49 |
| Technical Documentation | YES |
| JS tests | YES |
| Website | fancybirds.io |
| Timeline | 14 DECEMBER 2021 – 16 DECEMBER 2021 |
| Changelog | 16 DECEMBER 2021 – INITIAL AUDIT |

# Table of contents

# Introduction

AuditWhale (Consultant) was contracted by Fancy (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 December 14th, 2021 - December 16th,  2021.

# Scope

The scope of the project is smart contracts in the repository:
Repository:
      https://github.com/Fancy-Birds/staking
Commit:
      ea37a4b54e8fd81559361948ca7ed2e556a29e49
Technical Documentation: Yes,
https://docs.google.com/document/d/1vPhbIOP_WxmUvwRzO0tgGB-2KbHX5ejb
JS tests: Yes, in the repository
Contracts:
      FancyEscrowPool.sol
      FancyStakingPool.sol
      LiquidityMiningManager.sol View.sol
      base/BaseEscrowPool.sol
      base/AbstractRewards.sol
      base/TokenSaver.sol
      base/BasePool.sol
      interfaces/IBasePool.sol
      interfaces/IAbstractRewards.sol
      interfaces/ILiquidityMiningManager.sol
      interfaces/IFancyStakingPool.sol
      interfaces/IMintableBurnableERC20.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<br>▪ Ownership Takeover<br>▪ Timestamp Dependence<br>▪ Gas Limit and Loops<br>▪ DoS with (Unexpected) Throw<br>▪ DoS with Block Gas Limit<br>▪ Transaction-Ordering Dependence<br>▪ Style guide violation<br>▪ Costly Loop<br>▪ ERC20 API violation<br>▪ Unchecked external call<br>▪ Unchecked math<br>▪ Unsafe type inference<br>▪ Implicit visibility level<br>▪ Deployment Consistency<br>▪ Repository Consistency<br>▪ Data Consistency |
| Functional review | ▪ Business Logics Review<br>▪ Functionality Checks<br>▪ Access Control & Authorization<br>▪ Escrow manipulation<br>▪ Token Supply manipulation<br>▪ Assets integrity<br>▪ User Balances manipulation<br>▪ Data Consistency manipulation<br>▪ Kill-Switch Mechanism<br>▪ Operation Trails & Event Generation |

## Executive Summary

According to the assessment, the Customer's smart contracts are well-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 2 low severity issues.

## 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

No high severity issues were found.

■ ■ Medium

No medium severity issues were found.

■ Low

1. View functions iterate over or return an array of unpredictable size Contracts:

   FancyStakingPool.sol, FancyEscrowPool.sol, View.sol Functions: getTotalDeposit,

   getDepositsOf, fetchData

   Gas consumption grows with array size and starting from a certain size function could become inoperable.

   Recommendation: add(optional) *limit* and *offset* parameters for methods that return arrays of unpredictable size, store in state variable array total deposits for accounts to optimize function *getTotalDeposit*

2. State variables that could be declared constant

   Constant state variables should be declared constant to save gas.

   Contracts: LiquidityMiningManager.sol

   Variables: MAX_POOL_COUNT

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

## 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 2 low severity issues.

## Disclaimers

### AuditWhale  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.