Smart contract security audit

Falcon9inu

v.1.0

Table of Contents

1.0 Introduction

1.1 Project engagement

During November of 2021, Falcon9inu engaged CTDSec to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. Falcon9inu provided CTDSec with access to their code repository and whitepaper.

1.2 Disclaimer

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the network's fast-paced and rapidly changing environment, we at CTDSec recommend that Falcon9inu team put in place a bug bounty program to encourage further and active analysis of the smart contract.

2.0 Coverage

2.1 Target Code and Revision

For this audit, we performed research, investigation, and review of the Falcon9inu contract followed by issue reporting, along with mitigation and remediation instructions outlined in this report. The following code files are considered in-scope for the review:

Source:

https://bscscan.com/address/0x831742aD4f0B9f0b2A286036f8496aA773E69748#code

2.2 Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

№	Issue description.	Checking status
1	Compiler warnings.	PASSED
2	Race conditions and Reentrancy. Cross-function race conditions.	PASSED
3	Possible delays in data delivery.	PASSED
4	Oracle calls.	PASSED
5	Front running.	PASSED
6	Timestamp dependence.	PASSED
7	Integer Overflow and Underflow.	PASSED
8	DoS with Revert.	PASSED
9	DoS with block gas limit.	PASSED
10	Methods execution permissions.	PASSED
11	Economy model. If application logic is based on an incorrect economic model, the application would not function correctly and participants would incur financial losses. This type of issue is most often found in bonus rewards systems, Staking and Farming contracts, Vault and Vesting contracts, etc.	PASSED
12	The impact of the exchange rate on the logic.	PASSED
13	Private user data leaks.	PASSED
14	Malicious Event log.	PASSED
15	Scoping and Declarations.	PASSED
16	Uninitialized storage pointers.	PASSED

17	Arithmetic accuracy.	PASSED
18	Design Logic.	PASSED
19	Cross-function race conditions.	PASSED
20	Safe Zeppelin module.	PASSED
21	Fallback function security.	PASSED
22	Overpowered functions / Owner privileges	PASSED

3.0 Security Issues

3.1 High severity issues [0]

No high severity issues found.

3.2 Medium severity issues [0]

No medium severity issues found.

3.3 Low severity issues [2]

1. Catch statement unused

Fail try/catch statement is unused

```
try dividendTracker.setBalance(payable(from), balanceOf(from)) {} catch {}
try dividendTracker.setBalance(payable(to), balanceOf(to)) {} catch {}

if(!swapping) {
    uint256 gas = gasForProcessing;

    try dividendTracker.process(gas) returns (uint256 iterations, uint256 claims, uint256 lastProcessedIndex) {
        emit ProcessedDividendTracker(iterations, claims, lastProcessedIndex, true, gas, tx.origin);
    }
    catch {

    }
```

2. Missing error message

The function distributeBUSDDividends doesn't have any error message that will make it hard for users to understand what's happening.

```
function distributeBUSDDividends(uint256 amount) public onlyOwner{
  require(totalSupply() > 0);

  if (amount > 0) {
    magnifiedDividendPerShare = magnifiedDividendPerShare.add(
      (amount).mul(magnitude) / totalSupply()
    );
    emit DividendsDistributed(msg.sender, amount);

    totalDividendsDistributed = totalDividendsDistributed.add(amount);
  }
}
```

4.0 Summary of the audit

Contract is safe to be deployed.