Smart contract security audit

Egonswap

v.1.2

Table of Contents

1.0 Introduction **3**

1.1 Project engagement 3

1.2 Disclaimer 3

2.0 Coverage **4**

2.1 Target Code and Revision 4

2.2 Attacks made to the contract 5

3.0 Security Issues **7**

3.1 High severity issues [0] 7

3.2 Medium severity issues [2] 7

3.3 Low severity issues [0] 8

4.0 Summary of the audit **9**

1.0 Introduction

1.1 Project engagement

During August of 2021, Egonswap 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. Egonswap 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 Egonswap 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 Egonswap 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:

V1 review:

Eagleswap Contracts.zip [SHA256] -

2e64061c98de3099f8da4ab0a77b4e0bd0fbda1fd483cffbd6419db489bf8340

V2 review:

Egonswap Contracts.zip [SHA256] -

86456af5871a8717312e39ebd1a3fd0f587b5bda7f603f2c6c72906c93793f6c

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.	SOLVED BY DEV TEAM
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]

1.Reentrancyguard isn't applied to Masterchef contract.

```
contract MasterChef is Ownable {
    using SafeMath for uint256;
    using SafeBEP20 for IBEP20;

    // Info of each user.
    struct UserInfo {
        uint256 amount;         // How many LP tokens the user has provided.
        uint256 rewardDebt;     // Reward debt. See explanation below.
```

Recommendation:

We recommend reentrancyGuard, that will be used to prevent any possible reentrancy issues.

Instead:

```
contract MasterChef is Ownable, ReentrancyGuard {
    using SafeMath for uint256;
    using SafeBEP20 for IBEP20;

    // Info of each user.
    struct UserInfo {
        uint256 amount;         // How many LP tokens the user has provided.
        uint256 rewardDebt;     // Reward debt. See explanation below.
```

ReentrancyGuard was added in the contract, issue is fixed.

2.Withdraw/lp deposit functions without nonReentrant

```
function emergencyWithdraw(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
```

```
function withdraw(uint256 _pid, uint256 _amount) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
```

```
function deposit(uint256 _pid, uint256 _amount) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    updatePool( pid);
```

We recommend adding nonReentrant to the highlighted functions.

Functions are now protected with "nonReentrant" attribute.

3.3 Low severity issues [0]

No low severity issues found.

4.0 Summary of the audit

Contract has no issues and it’s safe to be deployed.