Security Assessment

OgeeSwap Farm

Jul 7th, 2021

Table of Contents

Summary

This report has been prepared for Ogee Finance to discover issues and vulnerabilities in the source code of the OgeeSwap Farm project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	OgeeSwap Farm
Platform	Heco
Language	Solidity
Codebase	https://github.com/ogeefinance/deployed-con/tree/master/
Commit	85a12074f2dbfec36adc5e3c395ec9e83b275378

Audit Summary

Delivery Date	Jul 07, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	Pending	Partially Resolved	Resolved	Acknowledged	Declined
Critical	0	0	0	0	0	0
Major	1	1	0	0	0	0
Medium	0	0	0	0	0	0
Minor	4	4	0	0	0	0
Informational	3	3	0	0	0	0
Discussion	0	0	0	0	0	0

Audit Scope

ID	file	SHA256 Checksum
MCE	farming/MasterChef.sol	45fa3cd817d008d4a4271fb53c10ad0acb1ea6e431c30f9e58d69196c3733996
FAR	farming	2c8628edf382f8eb089e49186de4dfb1af7fbdbc738dab0c58538f24f66f7951
OYT	farming/OYT.sol	40e20272244ff5407d5e5f828b0ace46a22520764f781ca1b8df5d0582700861

There is one depending injection contract in the current project:

`oyt` for the contract `MasterChef`.

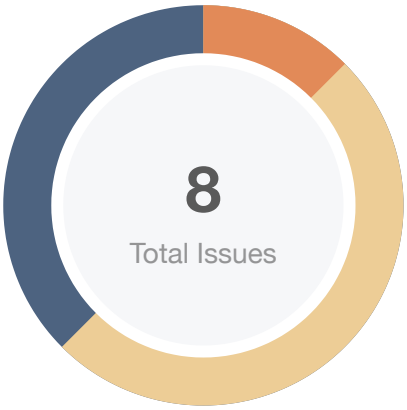
We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project.

To set up the project correctly, improve overall project quality and preserve upgradability, the following role is adopted in the codebase:

`owner` is adopted to update pool configurations, set up a new pool, and update the token emission rate for the contract `MasterChef`;

To improve the project's trustworthiness, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should also be considered to move to the execution queue of the `Timelock` contract.

Findings



Critical	**0** (0.00%)
Major	**1** (12.50%)
Medium	**0** (0.00%)
Minor	**4** (50.00%)
Informational	**3** (37.50%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
MCE-01	Function `add()` Not Being Restricted Properly	Volatile Code	Minor	Pending
MCE-02	Reentrancy Attack Risks	Logical Issue	Minor	Pending
MCE-03	Centralization Risks	**Centralization / Privilege**	**Minor**	**Pending**
MCE-04	Reentrancy Attack Risks	Logical Issue	Major	Pending
MCE-05	Incompatibility With Deflationary Tokens	Logical Issue	Minor	Pending
MCE-06	Lack of Event Emissions for Significant Transactions	Logical Issue	Informational	Pending
MCE-07	Potential Integer Overflow	Logical Issue	Informational	Pending
OYT-01	Potential Integer Overflow	Logical Issue	Informational	Pending

MCE-01 | Function `add()` Not Being Restricted Properly

Category	Severity	Location	Status
Volatile Code	Minor	farming/MasterChef.sol: 465	Pending

Description

The same LP token should not be added more than once to avoid potential incorrect rewards calculation.

Recommendation

We advise the client to detect whether the given pool for addition is a duplicate of an existing pool. The pool addition should be only allowed when no duplicate pool is detected. For instance, using a mapping of `addresses` -> `booleans` can restrict the same address being added twice.

MCE-02 | Reentrancy Attack Risks

Category	Severity	Location	Status
Logical Issue	Minor	farming/MasterChef.sol: 465, 482, 511, 518, 561, 611	Pending

Description

Function `MasterChef.updatePool()` has the following state updates after external calls of `aurora.mint()`:

```
530         oyt.mint(devaddr, oytReward.div(10));
531         oyt.mint(address(this), oytReward);
532         pool.accOytPerShare =
pool.accOytPerShare.add(oytReward.mul(1e12).div(lpSupply));
533         pool.lastRewardBlock = block.number;
```

We understand that this implementation is safe if the `oyt` token contract is implemented and injected properly by the client. However, given the fact that `oyt` is an external address, we must assume that the current implementation is vulnerable to reentrancy attack. In addition, functions `MasteChef.add()`, `MasterChef.set()`, `MasterChef.withdraw()` and `MasterChef.updateEmissionRate()` call function `MasterChef.updatePool()` directly or indirectly, which means these functions are vulnerable to reentrancy attack as well.

Recommendation

We advise the client to apply OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

MCE-03 | Centralization Risks

Category	Severity	Location	Status
Centralization / Privilege	Minor	farming/MasterChef.sol: 465, 482, 611	Pending

Description

The role `owner` is allowed to

- add a new pool by calling function `MasterChef.add()`;
- modify `allocPoint` and `depositFeeBP of a pool by calling function` MasterChef.set()`;
- modify the reward emission rate by calling function `MasterChef.updateEmissionRate()`.

Recommendation

We advise the client to handle the `owner` account carefully to avoid any potential hack. We also advise the client to consider the following solutions:

1. `Timelock` with reasonable latency for community awareness on privileged operations;
2. Multisig with community-voted 3rd-party independent co-signers;
3. DAO or Governance module increasing transparency and community involvement.

MCE-04 | Reentrancy Attack Risks

Category	Severity	Location	Status
Logical Issue	Major	farming/MasterChef.sol: 536, 561, 579	Pending

Description

The aforementioned functions emit events or update the state variables after the external call of function `pool.lpToken.safeTransfer()`. Therefore these functions are vulnerable to reentrancy attacks.

Recommendation

We advise the client to apply OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned function to prevent reentrancy attack.

MCE-05 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Logical Issue	Minor	farming/MasterChef.sol: 547, 572, 585	Pending

Description

Contract `MasterChef` updates user's balance after the user deposit token to the contract or withdraw token from the contract based on the input of token transfer. However, the input of token transfer does not always match the result of the token transfer. This fact might bring unexpected balance inconsistencies.

For example, a deflationary token AToken charges 10% of the transfer amount as transfer fees. If a user deposits 100 AToken to contract `MasterChef`, the contract will only receive 90 AToken, while `user.amount` is set to 100. When the user tries to withdraw the token from the contract, he will not be able to withdraw 100, which shows as `user.amount`, because, if he is the only person who deposited the contract, the contract only has 90 AToken.

Recommendation

We advise the client to carefully review the implementation of `MasterChef.poolInfo[_pid].lpToken` before setting a pool and ensure a deflationary token will not be used as `MasterChef.poolInfo[_pid].lpToken`.

MCE-06 | Lack of Event Emissions for Significant Transactions

Category	Severity	Location	Status
Logical Issue	Informational	farming/MasterChef.sol: 600, 605	Pending

Description

Functions `MasterChef.dev()` and `MasterChef.setFeeAddress()` involve significant transactions which would update the contract configurations. Missing event logs makes it difficult to track parameter or state changes.

Recommendation

We advise the client to emit events for the aforementioned functions.

MCE-07 | Potential Integer Overflow

Category	Severity	Location	Status
Logical Issue	Informational	farming/MasterChef.sol: 803	Pending

Description

`SafeMath.add()` is not used in the function `_writeCheckpoint()`, which might lead to integer overflow and cause potential incorrect processing result.

Recommendation

We advise the client to adopt `SafeMath.add()` to avoid potential integer overflow in the aforementioned function.

OYT-01 | Potential Integer Overflow

Category	Severity	Location	Status
Logical Issue	Informational	farming/OYT.sol: 489	Pending

Description

`SafeMath.add()` is not used in the function `_writeCheckpoint()`, which might lead to integer overflow and cause potential incorrect processing result.

Recommendation

We advise the client to adopt `SafeMath.add()` to avoid potential integer overflow in the aforementioned function.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.