Security Assessment

Pudding-Farm

May 11th, 2021

Summary

This report has been prepared for Pudding-Farm smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract 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	Pudding-Farm
Platform	Ethereum, Hoo Smart Chain
Language	Solidity
Codebase	https://github.com/Pudding-Finance/Pudding-Farm/commits/master/contracts/MasterChef.sol
Commits	e5149117d0c50f9a729a74420c96012fc1329a42

Audit Summary

Delivery Date	May 11, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Total Issues	8
Critical	0
Major	0
Medium	0
Minor	4
Informational	4
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
MCP	MasterChef.sol	38aea8db65ffa5dd526128a509e65ddb17b94e86e8539791defa177811e8da4c

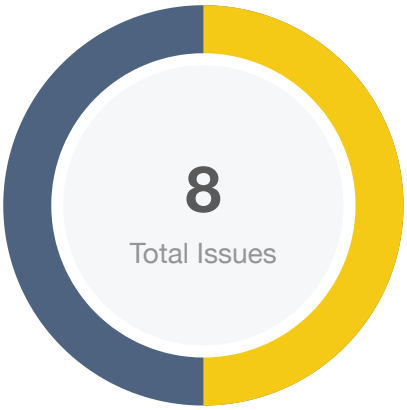
It should be noted that the system design includes some economic arguments and assumptions. These were explored to the extent that they clarified the intention of the code base, but we did not audit the mechanism design itself.

Additionally, financial models of blockchain protocols need to be resilient to attacks. It needs to pass simulations and verifications to guarantee the security of the overall protocol. The correctness of the financial model is not in the scope of the audit.

To bridge the trust gap between administrators and users, the administrator needs to express a sincere attitude with the consideration of the administrator team's anonymousness. The administrator has the responsibility to notify users with the following capability of the administrator:

- Managers can determine the manner and destination of lp token migration.

Findings



Critical	**0** (0.00%)
Major	**0** (0.00%)
Medium	**0** (0.00%)
Minor	**4** (50.00%)
Informational	**4** (50.00%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
MCP-01	Variable Naming Convention	Coding Style	Informational	Acknowledged
MCP-02	Missing Emit Events	Optimization	Informational	Acknowledged
MCP-03	add() Function Not Restricted	Volatile Code	Minor	Acknowledged
MCP-04	Recommended Explicit Pool Validity Checks	Logical Issue	Informational	Acknowledged
MCP-05	Incompatibility With Deflationary Tokens	Logical Issue	Minor	Acknowledged
MCP-06	Set `immutable` to Variables	Gas Optimization	Informational	Acknowledged
MCP-07	Over Minted Token	Logical Issue	Minor	Acknowledged
MCP-08	Unknown Implementation of migrator.migrate()	Logical Issue	Minor	Acknowledged

MCP-01 | Variable Naming Convention

Category	Severity	Location	Status
Coding Style	Informational	MasterChef.sol: 71	Acknowledged

Description

The linked variables do not conform to the standard naming convention of Solidity whereby local and state variable names should use mixedCase. In case the naming conventions are not followed, there should be proper documentation to explain the naming and the purpose of the variable.

Recommendation

According to the Solidity style guide, we advise that the naming conventions utilized by the linked statements be adjusted to reflect the correct type of declaration. The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

MCP-02 | Missing Emit Events

Category	Severity	Location	Status
Optimization	Informational	MasterChef.sol: 113~115, 165~167, 328~338	Acknowledged

Description

The function that affects the status of sensitive variables should be able to emit events as notifications to customers.

- `dev()`
- `setMigrator()`
- `updateMultiplier()`

Recommendation

Consider adding events for sensitive actions, and emit them in the function as follows.

```
event SetDev(address indexed user, address indexed _devaddr);

function dev(address _devaddr) public {
 require(msg.sender == devaddr, "dev: wut?");
 devaddr = _devaddr;
 emit SetDev(msg.sender, _devaddr);
}
```

MCP-03 | add() Function Not Restricted

Category	Severity	Location	Status
Volatile Code	Minor	MasterChef.sol: 123~136	Acknowledged

Description

The comment in line L122, mentioned // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do. The total amount of reward pudReward in function `updatePool()` will be incorrectly calculated if the same LP token is added into the pool more than once in function `add()`.

However, the comment behaviors are not reflected in the code as there isn't any valid restriction on preventing this issue. The current implementation is relying on the credibility of the owner to avoid repeatedly adding the same LP token to the pool, as the function will only be called by the owner.

Recommendation

Detect whether the given pool for addition is a duplicate of an existing pool. The pool addition is only successful when there is no duplicate. Using mapping of `addresses` -> `booleans`, which can restrict the same address being added twice.

Alleviation

The team responds that a LP token would never be added twice.

MCP-04 | Recommended Explicit Pool Validity Checks

Category	Severity	Location	Status
Logical Issue	Informational	MasterChef.sol: 139~149, 170~179, 187~198, 209~225, 228~247, 250~268, 312~320	Acknowledged

Description

There's no sanity check to validate if a pool is existent.

Recommendation

We advise the client to adopt following modifier `validatePool` to functions `set()`, `migrate()`, `pendingPudding()`, `updatePool()`, `deposit()`, `withdraw()` and `emergencyWithdraw()` as follows.

```
modifier validatePoolByPid(uint256 _pid) {
    require (_pid < poolInfo . length , "Pool does not exist") ;
    _;
}
```

MCP-05 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Logical Issue	Minor	MasterChef.sol: 228~247, 250~268	Acknowledged

Description

The MasterChef contract operates as the main entry for interaction with staking users. The staking users deposit LP tokens into the pool and in return get a proportionate share of the pool's rewards. Later on, the staking users can withdraw their own assets from the pool. In this procedure, `deposit()` and `withdraw()` are involved in transferring users' assets into (or out of) the protocol. When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged (and burned) transaction fee. As a result, this may not meet the assumption behind these low-level asset-transferring routines and will bring unexpected balance inconsistencies.

Recommendation

We advise the client to regulate the set of LP tokens supported in Pudding Farm and if there is a need to support deflationary tokens, add necessary mitigation mechanisms to keep track of accurate balances.

MCP-06 | Set `immutable` to Variables

Category	Severity	Location	Status
Gas Optimization	Informational	MasterChef.sol: 63, 65, 69, 82	Acknowledged

Description

The variables `pud`, `xPudding`, `pudPerBlock` and `startBlock` are only changed once in the `constructor()` function.

Recommendation

We advise the client to set `pud`, `xPudding`, `pudPerBlock` and `startBlock` as immutable variables.

MCP-07 | Over Minted Token

Category	Severity	Location	Status
Logical Issue	Minor	MasterChef.sol: 221	Acknowledged

Description

`updatePool()` function minted 100% + 10% (dev fee) of pudReward.

Recommendation

We advise the client to mint 100% of the calculated pudReward in total instead of 100% + 10%.

MCP-08 | Unknown Implementation of migrator.migrate()

Category	Severity	Location	Status
Logical Issue	Minor	MasterChef.sol: 165~167, 170~179	Acknowledged

Description

`setMigrator()` function can set the migrator contract to any contract that is implemented from `IMigratorChef` interface by the owner. As result, the invocation of `migrator.migrate()` in function `migrate()` may be as dangerous as it is unknown to the user. On the other hand, the project could lose the ability to upgrade and migrate if `setMigrator()` and `migrate()` are removed.

Recommendation

We would like to enquire about your precautions against abuse of the migrate functionality.

Alleviation

The team responds that it would be ensured there will be no abuse of the migrate functionality

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.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

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.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

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

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

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 string 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.