CERTIK

PancakeSwap

Security Assessment

October 13th, 2020

For:
PancakeSwap

Overview

Project Summary

Project Name	**PancakeSwap**
Description	A SushiSwap fork that attempts to differentiate itself by integrating YAM's / Compound's Governance mechanism.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 05e7fbdd16a94b7e18b51811eda411fb4a1b4b41 2. daf8da084170ec8fe7ff705a7c0489dc8f730e50

Audit Summary

Delivery Date	**October 13th, 2020**
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	1
Timeline	October 5th, 2020 - October 13th, 2020

Vulnerability Summary

Total Issues	**9**
Total Critical	0
Total Major	2
Total Medium	1
Total Minor	1
Total Informational	5

Executive Summary

The PancakeSwap implementation is a fork of the SushiSwap implementation that introduces a new pool reserved in the first position of the pool array yielding rewards proportionate to the rest of the liquidity in the contract. We identified some major and medium severity flaws that were dealt with by the PancakeSwap team, however a new vulnerability emerged due to the fix for the `addressList` inaccuracy finding that should be tended to as soon as possible.

Files In Scope

ID	Contract	Location
MCF	MasterChef.sol	contracts/MasterChef.sol
SCF	SousChef.sol	contracts/SousChef.sol
SBR	SyrupBar.sol	contracts/SyrupBar.sol

Findings

ID	Title	Type	Severity	Resolved
MCF-01	Variable Naming Convention	Coding Style	Informational	!
MCF-02	Comment Typo	Coding Style	Informational	✓
MCF-03	Centralized Control of Bonus Multiplier	Logical Issue	Informational	✓
MCF-04	Assignment Optimization	Gas Optimization	Informational	✓
SBR-01	Incorrect Delegation Flow	Logical Issue	Major	✓
SBR-02	Inexistent Delegate Transfer	Logical Issue	Medium	✓
SCF-01	`addressList` Inaccuracy	Logical Issue	Minor	✓
SCF-02	Contract Purpose Unclear	Logical Issue	Informational	✓
SCF-03	Incorrect Reset Mechanism	Logical Issue	Major	✓

MCF-01: Variable Naming Convention

Type	Severity	Location
Coding Style	Informational	MasterChef.sol L71

Description:

The linked variables do not conform to the standard naming convention of Solidity whereby functions and variable names utilize the `camelCase` format, unless variables are declared as `constant` in which case they utilize the `UPPER_CASE` format.

Recommendation:

We advise that the naming conventions utilized by the linked statements are adjusted to reflect the correct type of declaration according to the Solidity style guide.

Alleviation:

The PancakeSwap development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase due to time constraints.

MCF-02: Comment Typo

Type	Severity	Location
Coding Style	Informational	MasterChef.sol L79

Description:

The linked comment statement contains a typo in its body, namely `poitns`.

Recommendation:

We advise that the comment text is corrected.

Alleviation:

The comment typo was properly fixed.

MCF-03: Centralized Control of Bonus Multiplier

Type	Severity	Location
Logical Issue	Informational	MasterChef.sol L113-L115

Description:

The function `updateMultiplier` can alter the `BONUS_MULTIPLIER` variable and consequently the output of `getMultiplier` which is directly utilized for the minting of new cake tokens.

Recommendation:

This is intended functionality of the protocol, however users should be aware of this functionality.

Alleviation:

The PancakeSwap team informed us that there is a 6 hour timelock on the MasterChef contract with regards to all pool reward changes which are also first voted on by SYRUP holders through their voting portal using a Snapshot mechanism. This decentralizes the aspect of changing the multipliers via governance by SYRUP holders.

MCF-04: Assignment Optimization

Type	Severity	Location
Gas Optimization	Informational	MasterChef.sol L143

Description:

The linked statement will only yield a different output stored to `totalAllocPoint` only if the condition of L146 yields `true`.

Recommendation:

As a result of the above, it is more optimal to move the assignment of L143 to the `if` block of L146.

Alleviation:

The assignment was properly moved to the linked `if` block, optimizing the code segment.

SBR-01: Incorrect Delegation Flow

Type	Severity	Location
Logical Issue	Major	SyrupBar.sol L17

Description:

Whenever new SYRUP tokens are minted, new delegates are moved from the zero address to the recipient of the minting process. However, whenever tokens are burned, new delegates are once again moved from the zero address to the recipient whereas delegates should be moved on the opposite way.

Recommendation:

We advise that the `address(0)` and `_from` variable orders are swapped on L17 to alleviate this issue. At its current state, it breaks the delegate mechanism and can also lead to a user being unable to mint / burn tokens in case the upper limit of a `uint256` is reached due to the `SafeMath` utilization on L233.

Alleviation:

The delegation flow was fixed in the source code of the GitHub repository, however the issue still persists in the deployed version of PancakeSwap. However, the `SYRUP` token will not be utilized for the DAO governance by the PancakeSwap team.

SBR-02: Inexistent Delegate Transfer

Type	Severity	Location
Logical Issue	Medium	SyrupBar.sol L1

Description:

The `transfer` and `transferFrom` functions of the YAM project transfer delegates as well via overridence. The PancakeSwap implementation does not, leading to an inconsistency in the delegates of each `address`.

Recommendation:

We advise that the `transfer` and `transferFrom` functions are properly overriden to also transfer delegates on each invocation from the sender of the funds to the recipient.

Alleviation:

After evaluating with PancakeSwap, we came to the conclusion that this functionality is unnecessary as delegates are not and will not be utilized in any form of DAO governance mechanism.

SCF-01: `addressList` Inaccuracy

Type	Severity	Location
Logical Issue	Minor	SousChef.sol L120-L122, L148-L154

Description:

The first linked `if` block pushes a new `address` to the `addressList` array in the case the `userInfo` mapping lookup yields `0` on the `amount` member. This case is possible even after the user has already been added to the array, either by invoking `emergencyWithdraw` or withdrawing the full amount held by the user.

Recommendation:

We advise that the `push` mechanism is revised to ensure that the user does not already exist in the array.

Alleviation:

The PancakeSwap team altered the condition for pushing new items to the `addressList` array, however duplicates can still exist. After conversing with the team, we were informed that the array is not utilized on-chain and is meant to aid off-chain processes in an airdrop mechanism which will eliminate duplicate addresses. As such, this issue can be safely ignored. We would like to note that this is not an optimal mechanism to conduct this, as it would be better to instead rely on emitted `event`s and blockchain analysis rather than contract storage.

SCF-02: Contract Purpose Unclear

Type	Severity	Location
Logical Issue	Informational	SousChef.sol L1-L156

Description:

The `SousChef` contract tracks a reward schedule based on the deposited SYRUP tokens, however the variables of the `UserInfo` struct are never actually utilized to provide any reward.

Recommendation:

We advise that further documentation is produced that details the purpose of the contract, as it should seemingly interoperate with another contract that reads data from it.

Alleviation:

The purpose of the contract is for SYRUP holders to stake their tokens and accumulate rewards on paper rather than on-chain which will be then distributed by the PancakeSwap team. As such, we believe that the purpose of the contract has been sufficiently described. We would like to note that this type of distribution of rewards purely relies on the honesty of PancakeSwap and does not utilize any on-chain or decentralized mechanisms.

SCF-03: Incorrect Reset Mechanism

Type	Severity	Location
Logical Issue	Major	SousChef.sol L148-L154

Description:

The `emergencyWithdraw` function is meant to "reset" a user's state and withdraw his deposited tokens. In this case, the `rewardPending` variable of the `user` struct is not zeroed out.

Recommendation:

As the `rewardPending` member is cumulative, it is possible to exploit this behavior and artificially increase the pending rewards of a user. We advise that either a manual `0` assignment statement is introduced in the `emergencyWithdraw` function or a `delete` operation is conducted on the full struct located at `userInfo[msg.sender]`.

Alleviation:

The `emergencyWithdraw` function was properly fixed to zero out all members of the `UserInfo` struct.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that 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 exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that 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 and 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.