DWEBOX.COM

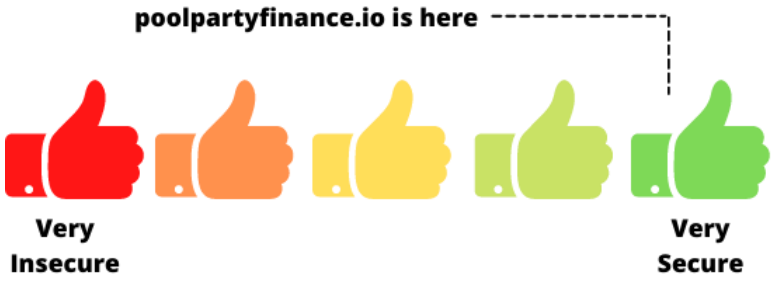
Smart contract code security analysis - AUDIT

Company: poolpartyfinance.io

Date: 06/August/2021

Dwebox.com was contacted by Poolpartyfinance.io (“CLIENT”) to conduct the smart contract code security analysis. This report presents all findings and vulnerabilities from the code review of the Customer’s smart contracts.

EXECUTIVE SUMMARY



Poolpartyfinance.io is a Goosedefi fork, providing dex services (decentralized exchange) through PancakeswapV2 , and implementing farms/pools yielding rewards proportionate to the rest of the liquidity in the contract.

Our team performed a full analysis of the code, manual audit and automated checks using Slither. All issues found during the analysis have been manually reviewed by Dwebox team. Security engineers found **0 majors**, **1 medium, 1 very low** and **3 informational** issues during the audit of the smart contracts.

Disclaimer: the audit scope is limited to the files provided. We cannot guarantee the security of the smart contracts that are not in the scope.

Files in Scope

pool_masterchef.sol created on August 0, 2021

poolpartymsc _token.sol created on May 21, 2021

pool _masterchef.sol has been released at **0xEC80E60db73C5fd82c8184099027985af5d1Ff93**

poolpartymsc _token has been released at **0x9c9ac8b098a7d47ed1834599ce2dc29cb94103e9**

Solidity versions used: **^0.6.12 for masterchef, ^0.4.24 for token**

Platform: EVM Chain on Binance Smart Chain (network id 56)

BEP-20 CONFORMANCE

This test is used to check for conformance of token with BEP-20 standard:

- Functions return correct type
- View functions are view
- All events are present (core events: Transfer & Approve)
- All functions are present (core functions: all)
- Functions emit events
- Events' parameters are indexed correctly

Function Name	Present?	Type?	Correct return value?	Event type?	Passed?
symbol	Function is **present**	Function is of type **view**	Function returns **correct value**	-	Passed all tests
name	Function is **present**	Function is of type **view**	Function returns **correct value**	-	Passed all tests
totalSupply	Function is **present**	Function is of type **view**	Function returns **correct value**	-	Passed all tests
balanceOf(address)	Function is **present**	Function is of type **view**	Function returns **correct value**	-	Passed all tests
allowance(address, address)	Function is **present**	Function is of type **view**	Function returns **correct value**	-	Passed all tests
transfer(address, uint256)	Function is **present**	Function is of type **external**	Function returns **correct value**	**Transfer**	Passed all tests
transferFrom(address, address, uint256)	Function is **present**	Function is of type **external**	Function returns **correct value**	**Transfer**	Passed all tests

approve(address, uint256)	Function is **present**	Function is of type **external**	Function returns **correct value**	**Approval**	Passed all tests

RISK AND SEVERITY

Dwebox.com uses 5 levels of severity:

- Very Secure: lowest level vulnerability, information statements that do not affect the smart contract execution and code style violation.
- Secure: Outdated or unused code that will not have impact on execution during runtime
- Somewhat secure: Vulnerabilities that should be fixed but their exploit will not lead to data manipulation, violate the business case of the smart contract or result in any asset loss.
- Not secure: Vulnerabilities that may be difficult to exploit but have vital important and significance in the smart contract execution. Such example of functions with public access.
- Very insecure: Vulnerability that is exploitable by a hacker with malicious intents and will lead to asset loss.

VN1 –
VARIABLE NAMING

INFORMATIONAL – VERY SECURE

Some of the variables inside the smart contract do not conform with coding style, standard naming convention in Solidity as some functions and variables must use camelCase (variableName) format. Unless variables are declared “constant” based on Solidity standard where they would user UPPER_CASE (VARIABLE_NAME) format.

We advise on changing naming conventions to utilized the correct type of declaration based on Solidity coding style. Overall this is does not impose a threat to the smart contract and its business operations.

VN2 - COMMENT TYPO

INFORMATIONAL – VERY SECURE

On eyeballing inspection of all comments in all smart contracts (pool_masterchef.sol, poolpartymsc _token.sol) a typo can be found on line 1290 of pool _masterchef.sol. The name “EGG” should be changed to “CoinToken” or “MSC”.

We advise correcting the typo. Overall this is does not impose a threat to the smart contract and its business operations.

VN3 - BONUS MULTIPLIER CENTRALIZATION

INFORMATIONAL – VERY SECURE

The smart contract pool _masterchef.sol implements the function
` updatePool` which can change the BONUS_MULTIPLIER variable that may affect rewards of farms / pools. The output of getMultiplier is directly utilized in minting tokens and create inflation

We advise keeping this functionality if it is intended of the protocol but inform users of poolpartyfinance.io

VN4 - VOLATILE CODE

VERY LOW – SECURE

The smart contract pool_masterchef.sol implements lpToken parameter and safeTransfer() function which are arbitrarily complex. This may lead or could have a malicious implementation which calls function deposit(). This can result in the invocation of safeTransfer() before updating user.amount variable thus miscalculating users' balance being displayed.

We advise following check-effects-interaction principle for best practices and update user.amount before safeTransfer().

VN5 - POOLS

MEDIUM – SOMEWHAT SECURE

The original masterchef of the original fork Goosedefi implements implements a timelock on the smart contract, meaning that users will get to know of upcoming changes at least 6 hours before it happens. Considering that the updateEmissionRate has a cap of 25 (max) this is not a major problem.

We advise keeping this functionality if it is intended of the protocol but would advise implementing a timelock smart contract.

VULNERABILITY CHECKLIST – MASTERCHEF

✓ = safe ✗ = not safe

Address

✓ Re-Entrancy
✓ Timestamp Dependency
✓ Transaction-Ordering Dependency
✓ Parity Multisig Bug
✓ Callstack Depth Attack
✓ Integer Overflow & Underflow

BEP20

✓ Re-Entrancy
✓ Timestamp Dependency
✓ Transaction-Ordering Dependency
✓ Parity Multisig Bug
✓ Callstack Depth Attack
✓ Integer Overflow & Underflow

Token

✓ Re-Entrancy
✓ Timestamp Dependency
✓ Transaction-Ordering Dependency
✓ Parity Multisig Bug
✓ Callstack Depth Attack
✓ Integer Overflow & Underflow

Context

✓ Re-Entrancy
✓ Timestamp Dependency
✓ Transaction-Ordering Dependency
✓ Parity Multisig Bug
✓ Callstack Depth Attack
✓ Integer Overflow & Underflow

VULNERABILITY CHECKLIST – TOKEN

✓ = safe ✗ = not safe

Address

✓ Re-Entrancy
✓ Timestamp Dependency
✓ Transaction-Ordering Dependency
✓ Parity Multisig Bug
✓ Callstack Depth Attack
✓ Integer Overflow & Underflow

BEP20

✓ Re-Entrancy
✓ Timestamp Dependency
✓ Transaction-Ordering Dependency
✓ Parity Multisig Bug
✓ Callstack Depth Attack
✓ Integer Overflow & Underflow

Token

✓ Re-Entrancy
✓ Timestamp Dependency
✓ Transaction-Ordering Dependency
✓ Parity Multisig Bug
✓ Callstack Depth Attack
✓ Integer Overflow & Underflow

Context

✓ Re-Entrancy
✓ Timestamp Dependency
✓ Transaction-Ordering Dependency
✓ Parity Multisig Bug
✓ Callstack Depth Attack
✓ Integer Overflow & Underflow

VULNERABILITY CHECKLIST – TOKEN CONTINUED

✓ = safe ✗ = not safe

Ownable

✓ Re-Entrancy
✓ Timestamp Dependency
✓ Transaction-Ordering Dependency
✓ Parity Multisig Bug
✓ Callstack Depth Attack
✓ Integer Overflow & Underflow