RD AUDITORS

PulseDAO, Contract Code Review and Security Analysis Report

Customer: PulseDAO
Prepared on: 31st October 2023
Platform: Binance
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer

This document may contain confidential information about its systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities are fixed - upon the decision of the customer.

Document

Name	Smart Contract Code Review and Security Analysis Report of PulseDAO
Platform	PulseChain/ Solidity
File	TokenVault.sol
MD5 hash	7ff1741fd3562e93381d101d151f2d05
SHA256 hash	5f8b7ff44e0c7fe0cfcaeb940fc2ab5e75e4be5252ec4a4b7ac603c0192d1be5
File	PlsVault.sol
MD5 hash	d1ee81f2a85a5a632587bab372fdd620
SHA256 hash	a06e4d2aa40fcdc2345044a6fc9c04d5a76e2c8fdaaf8cbc1ef0be398c1d3eef
Date	31/10/2023

Introduction

RD Auditors (Consultant) were contracted by PulseDAO (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report represents the findings of the security assessment of the customer's smart contract and its code review conducted between 26th - 31st October 2023.

This contract consists of two files.

Project Scope

The scope of the project is a smart contract. We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, the customer's solidity smart contract is now **Poorly-Secured.**

Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, the manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found the following;

Total Issues	5
Critical	5
High	0
Medium	0
Low	0
Very Low	0

Code Quality

The libraries within this smart contract are part of a logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned to a specific address and its properties/methods can be reused many times by other contracts.

The PulseDAO team has not provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is almost commented. Commenting can provide rich documentation for functions, return variables and more. Use of the Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given the PulseDAO code as a Github link:

https://github.com/decentralizeX/PulseDAO/blob/main/contracts/pulse-ecosystem/tokenVault.sol

https://github.com/decentralizeX/PulseDAO/blob/main/contracts/pulse-ecosystem/plsVault.sol

The hash of that file is mentioned in the table. As mentioned above, it's recommended to write comments on smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects and even core code blocks that are written well and systematically.

AS-IS Overview

PulseDAO .sol

File And Function Level Report

Contract: TokenVault

Inherit: ReentrancyGuard

Import: IERC20, SafeERC20, ReentrancyGuard, IGovernor, IMasterchef, IacPool

Observation: Not Passed

Test Report: Not Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	deposit	write	Passed	All Passed	No Issue	Passed
2	harvest	write				
3	withdraw	write	Passed	All Passed	No Issue	Passed
4	selfHarvest	write	Passed	All Passed	No Issue	Passed
5	collectCommission	write	Passed	All Passed	No Issue	Passed
6	collectCommissionAuto	write	Passed	All Passed	No Issue	Passed
7	updateFees	write	Not Passed	Not Passed	Need to check	Not Passed
8	updateTreasury	write	Not Passed	Not Passed	Need to check	Not Passed
9	WithdrawStuckTokens	write	Not Passed	Not Passed	Need to check	Not Passed

10	setMasterchef Address	write	Passed	All Passed	No Issue	Passed
11	SetPoolPayout	write	Passed	All Passed	No Issue	Passed
12	UpdateSettings	write	Passed	All Passed	No Issue	Passed
13	ViewStakeEarnings	read	Passed	All Passed	No Issue	Passed
14	ViewUserTotalEarnings	read	Passed	All Passed	No Issue	Passed
15	multicall	read	Passed	All Passed	No Issue	Passed
16	CalculateTotalPendingDTXRewards	read	Passed	All Passed	No Issue	Passed
17	ViewPoolPayout	read	Passed	All Passed	No Issue	Passed
18	ViewPoolMinServe	read	Passed	All Passed	No Issue	Passed
19	getNrOfStakes	read	Passed	All Passed	No Issue	Passed
20	PublicBalanceOf	read	Passed	All Passed	No Issue	Passed
21	VirtualAccDtxPerShare	write	Passed	All Passed	No Issue	Passed
22	PayFee	write	Passed	All Passed	No Issue	Passed

Contract: PlsVault

Inherit: ReentrancyGuard

Import: IERC20, SafeERC20, ReentrancyGuard, IGovernor, IMasterchef ,IacPool

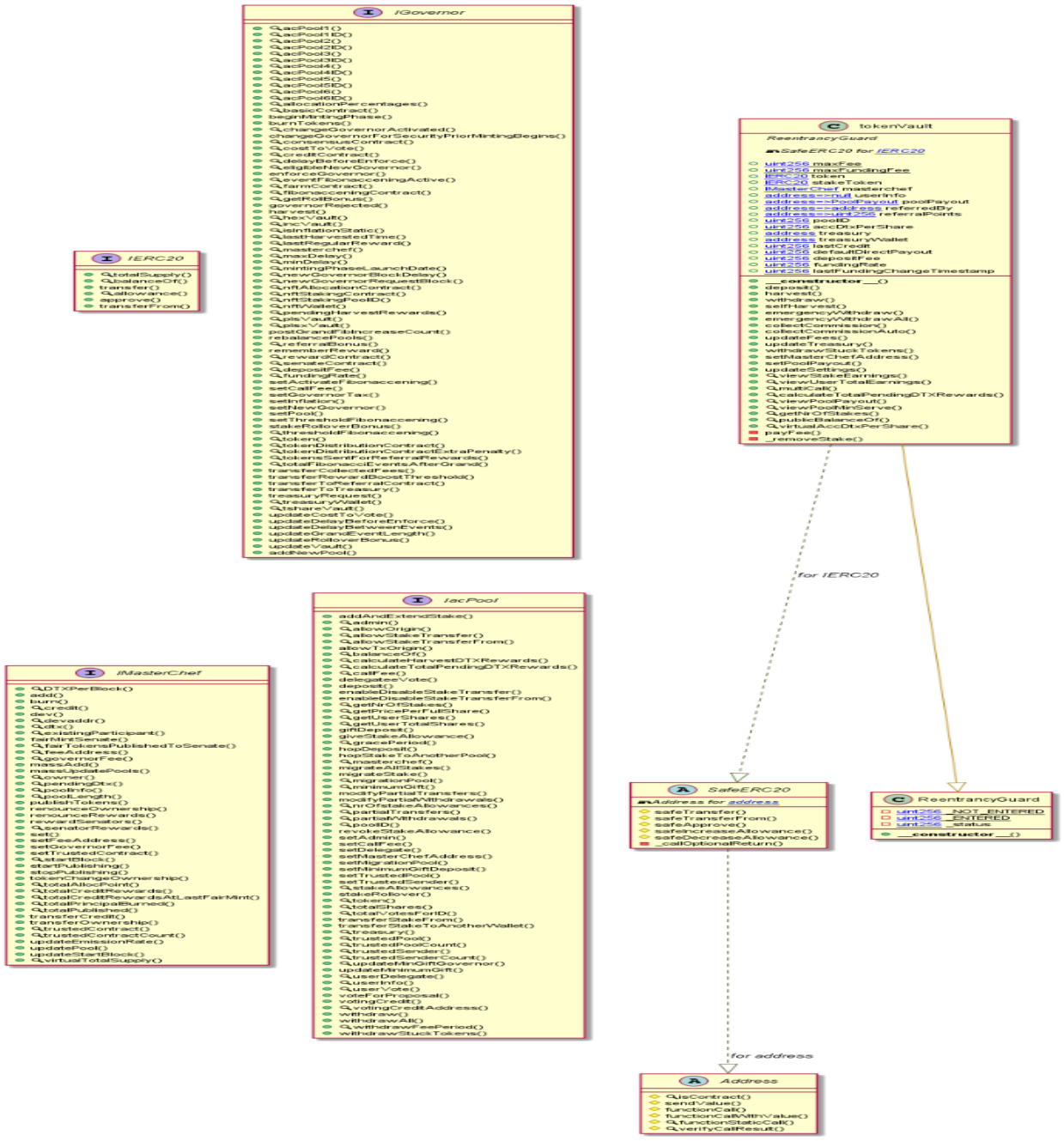
Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	deposit	write	Passed	All Passed	No Issue	Passed
2	harvest	write	Passed	All Passed	No Issue	Passed
3	withdraw	write	Passed	All Passed	No Issue	Passed
4	selfHarvest	write	Passed	All Passed	No Issue	Passed
5	emergencyWithdraw	write	Passed	All Passed	No Issue	Passed
6	emergencyWithdrawAll	write	Passed	All Passed	No Issue	Passed
7	CollectCommissionAuto	write	Passed	All Passed	No Issue	Passed
8	UpdateFee	write	Not Passed	Not Passed	Need to check	Not Passed
9	UpdateTreasury	write	Not Passed	Not Passed	Need to check	Not Passed
10	withdrawStuckTokens	write	Not Passed	Not Passed	Need to check	Not Passed
11	SetMasterchefAddress	write	Passed	All Passed	No Issue	Passed
12	setPoolPayout	write	Passed	All Passed	No Issue	Passed
13	UpdateSettings	write	Passed	All Passed	No Issue	Passed
14	ViewStakeEarnings	read	Passed	All Passed	No Issue	Passed
15	ViewUserTotalEarnings	read	Passed	All Passed	No Issue	Passed
16	multiCall	read	Passed	All Passed	No Issue	Passed
17	CalculateTotalPendingDTXRewards	read	Passed	All Passed	No Issue	Passed
18	ViewPoolPayout	read	Passed	All Passed	No Issue	Passed
19	ViewPoolMinServe	read	Passed	All Passed	No Issue	Passed

20	getNrOfStakes	read	Passed	All Passed	No Issue	Passed
21	publicBalanceOf	read	Passed	All Passed	No Issue	Passed
22	VirtualACCDtxPerShare	read	Passed	All Passed	No Issue	Passed
23	PayFee	write	Passed	All Passed	No Issue	Passed
24	_removeStake	write	Passed	All Passed	No Issue	Passed
25	_harvest	write	Passed	All Passed	No Issue	Passed

Code Flow Diagram - PulseDAO



Code Flow Diagram - Slither Results Log

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in tokenVault.harvest(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 727:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in tokenVault.updateTreasury(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 878:1:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 995:25:

Gas costs:

Gas requirement of function tokenVault.selfHarvest is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 783:1:

Gas costs:

Gas requirement of function tokenVault.collectCommission is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 841:1:

Gas costs:

Gas requirement of function tokenVault.collectCommissionAuto is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 850:1:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

more

Pos: 790:8:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

more

Pos: 843:3:

Constant/View/Pure functions:

IacPool.setAdmin() : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 517:4:

Similar variable names:

tokenVault._removeStake(address,uint256) : Variables have very similar names "stakes" and "_staker". Note: Modifiers are currently not considered by this static analysis.
Pos: 1020:12:

No return:

IacPool.withdrawFeePeriod(): Defines a return type but never explicitly returns a value.
Pos: 607:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 916:2:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 998:17:

Solidity Static Analysis

```
tokenVault.setMasterChefAddress(IMasterChef,uint256) (tokenVault.sol#1129-1132) should emit an event for:
        - poolID = _newPoolID (tokenVault.sol#1131)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Address.functionCallWithValue(address,bytes,uint256,string) (tokenVault.sol#60-71) has external calls inside a loop: (success,r
eturndata) = target.call{value: value}(data) (tokenVault.sol#69)
tokenVault.virtualAccDtxPerShare() (tokenVault.sol#1204-1207) has external calls inside a loop: _pending = IMasterChef(masterch
ef).pendingDtx(poolID) (tokenVault.sol#1205)
tokenVault.virtualAccDtxPerShare() (tokenVault.sol#1204-1207) has external calls inside a loop: (accDtxPerShare + _pending * 1e
12 / stakeToken.balanceOf(address(this))) (tokenVault.sol#1206)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Reentrancy in tokenVault.deposit(uint256,address) (tokenVault.sol#927-951):
        External calls:
        - harvest() (tokenVault.sol#929)
                - IMasterChef(masterchef).updatePool(poolID) (tokenVault.sol#957)
        - stakeToken.safeTransferFrom(msg.sender,address(this),_amount) (tokenVault.sol#930)
        State variables written after the call(s):
        - referredBy[msg.sender] = referral (tokenVault.sol#933)
Reentrancy in tokenVault.deposit(uint256,address) (tokenVault.sol#927-951):
        External calls:
        - harvest() (tokenVault.sol#929)
                - IMasterChef(masterchef).updatePool(poolID) (tokenVault.sol#957)
        - stakeToken.safeTransferFrom(msg.sender,address(this),_amount) (tokenVault.sol#930)
        - stakeToken.safeTransfer(treasuryWallet,_depositFee) (tokenVault.sol#941)
        State variables written after the call(s):
        - userInfo[msg.sender].push(UserInfo(_amount,_debt,_depositFee,block.timestamp)) (tokenVault.sol#946-948)
```

```
Reentrancy in tokenVault.harvest() (tokenVault.sol#956-962):
        External calls:
        - IMasterChef(masterchef).updatePool(poolID) (tokenVault.sol#957)
        State variables written after the call(s):
        - accDtxPerShare += _accumulatedRewards * 1e12 / stakeToken.balanceOf(address(this)) (tokenVault.sol#961)
        - lastCredit = _currentCredit (tokenVault.sol#960)
Reentrancy in tokenVault.selfHarvest(uint256[],address) (tokenVault.sol#1012-1046):
        External calls:
        - harvest() (tokenVault.sol#1015)
                - IMasterChef(masterchef).updatePool(poolID) (tokenVault.sol#957)
        - payFee(user[_stakeID[i]],msg.sender) (tokenVault.sol#1021)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (tokenVault.sol#188)
                - (success,returndata) = target.call{value: value}(data) (tokenVault.sol#69)
                - stakeToken.safeTransfer(treasuryWallet,commission) (tokenVault.sol#1230)
        - IMasterChef(masterchef).publishTokens(msg.sender,_payout) (tokenVault.sol#1027)
        - IMasterChef(masterchef).publishTokens(address(this),_payout) (tokenVault.sol#1031)
        - IacPool(_harvestInto).giftDeposit(_payout,msg.sender,poolPayout[_harvestInto].minServe) (tokenVault.sol#1032)
        External calls sending eth:
        - payFee(user[_stakeID[i]],msg.sender) (tokenVault.sol#1021)
                - (success,returndata) = target.call{value: value}(data) (tokenVault.sol#69)
        State variables written after the call(s):
        - referralPoints[msg.sender] += _payout (tokenVault.sol#1036)
        - referralPoints[referredBy[msg.sender]] += _payout (tokenVault.sol#1037)
```

```
Reentrancy in tokenVault.withdraw(uint256,address) (tokenVault.sol#968-1009):
        External calls:
        - harvest() (tokenVault.sol#969)
                - IMasterChef(masterchef).updatePool(poolID) (tokenVault.sol#957)
        - payFee(user,msg.sender) (tokenVault.sol#975)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (tokenVault.sol#188)
                - (success,returndata) = target.call{value: value}(data) (tokenVault.sol#69)
                - stakeToken.safeTransfer(treasuryWallet,commission) (tokenVault.sol#1230)
        - IMasterChef(masterchef).publishTokens(msg.sender,_toWithdraw) (tokenVault.sol#986)
        - IMasterChef(masterchef).publishTokens(address(this),_toWithdraw) (tokenVault.sol#991)
        - IacPool(_harvestInto).giftDeposit(_toWithdraw,msg.sender,poolPayout[_harvestInto].minServe) (tokenVault.sol#992)
        External calls sending eth:
        - payFee(user,msg.sender) (tokenVault.sol#975)
                - (success,returndata) = target.call{value: value}(data) (tokenVault.sol#69)
        State variables written after the call(s):
        - referralPoints[msg.sender] += _toWithdraw (tokenVault.sol#996)
        - referralPoints[referredBy[msg.sender]] += _toWithdraw (tokenVault.sol#997)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in tokenVault.payFee(tokenVault.UserInfo,address) (tokenVault.sol#1209-1238):
        External calls:
        - stakeToken.safeTransfer(treasuryWallet,commission) (tokenVault.sol#1230)
        Event emitted after the call(s):
        - CollectedFee(_userAddress,commission) (tokenVault.sol#1236)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

```
tokenVault.payFee(tokenVault.UserInfo,address) (tokenVault.sol#1209-1238) uses timestamp for comparisons
        Dangerous comparisons:
        - lastFundingChangeTimestamp > _lastAction (tokenVault.sol#1214)
        - secondsSinceLastaction >= 3600 && fundingRate > 0 (tokenVault.sol#1221)
        - commission > user.amount * 2 / 10 (tokenVault.sol#1226)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Address.verifyCallResult(bool,bytes,string) (tokenVault.sol#91-109) uses assembly
        - INLINE ASM (tokenVault.sol#101-104)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

ReentrancyGuard.nonReentrant() (tokenVault.sol#827-839) has costly operations inside a loop:
        - _status = _ENTERED (tokenVault.sol#832)
ReentrancyGuard.nonReentrant() (tokenVault.sol#827-839) has costly operations inside a loop:
        - _status = _NOT_ENTERED (tokenVault.sol#838)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop

Address.functionCall(address,bytes) (tokenVault.sol#40-42) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (tokenVault.sol#52-58) is never used and should be removed
Address.functionStaticCall(address,bytes) (tokenVault.sol#73-75) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (tokenVault.sol#77-86) is never used and should be removed
Address.sendValue(address,uint256) (tokenVault.sol#33-38) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (tokenVault.sol#140-153) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (tokenVault.sol#164-175) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (tokenVault.sol#155-162) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
Pragma version0.8.20 (tokenVault.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0
.8.16
solc-0.8.20 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (tokenVault.sol#33-38):
        - (success) = recipient.call{value: amount}() (tokenVault.sol#36)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (tokenVault.sol#60-71):
        - (success,returndata) = target.call{value: value}(data) (tokenVault.sol#69)
Low level call in Address.functionStaticCall(address,bytes,string) (tokenVault.sol#77-86):
        - (success,returndata) = target.staticcall(data) (tokenVault.sol#84)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
Function IMasterChef.DTXPerBlock() (tokenVault.sol#413) is not in mixedCase
Parameter IMasterChef.updateEmissionRate(uint256)._DTXPerBlock (tokenVault.sol#517) is not in mixedCase
Contract tokenVault (tokenVault.sol#845-1255) is not in CapWords
Parameter tokenVault.deposit(uint256,address)._amount (tokenVault.sol#927) is not in mixedCase
Parameter tokenVault.withdraw(uint256,address)._stakeID (tokenVault.sol#968) is not in mixedCase
Parameter tokenVault.withdraw(uint256,address)._harvestInto (tokenVault.sol#968) is not in mixedCase
Parameter tokenVault.selfHarvest(uint256[],address)._stakeID (tokenVault.sol#1012) is not in mixedCase
Parameter tokenVault.selfHarvest(uint256[],address)._harvestInto (tokenVault.sol#1012) is not in mixedCase
Parameter tokenVault.emergencyWithdraw(uint256)._stakeID (tokenVault.sol#1049) is not in mixedCase
Parameter tokenVault.collectCommission(address[],uint256[][])._beneficiary (tokenVault.sol#1070) is not in mixedCase
Parameter tokenVault.collectCommission(address[],uint256[][])._stakeID (tokenVault.sol#1070) is not in mixedCase
Parameter tokenVault.collectCommissionAuto(address[])._beneficiary (tokenVault.sol#1079) is not in mixedCase
Parameter tokenVault.withdrawStuckTokens(address)._tokenAddress (tokenVault.sol#1117) is not in mixedCase
Parameter tokenVault.setMasterChefAddress(IMasterChef,uint256)._masterchef (tokenVault.sol#1129) is not in mixedCase
Parameter tokenVault.setMasterChefAddress(IMasterChef,uint256)._newPoolID (tokenVault.sol#1129) is not in mixedCase
Parameter tokenVault.setPoolPayout(address,uint256,uint256)._poolAddress (tokenVault.sol#1137) is not in mixedCase
Parameter tokenVault.setPoolPayout(address,uint256,uint256)._amount (tokenVault.sol#1137) is not in mixedCase
Parameter tokenVault.setPoolPayout(address,uint256,uint256)._minServe (tokenVault.sol#1137) is not in mixedCase
Parameter tokenVault.updateSettings(uint256)._defaultDirectHarvest (tokenVault.sol#1144) is not in mixedCase
Parameter tokenVault.viewStakeEarnings(address,uint256)._user (tokenVault.sol#1149) is not in mixedCase
Parameter tokenVault.viewStakeEarnings(address,uint256)._stakeID (tokenVault.sol#1149) is not in mixedCase
Parameter tokenVault.viewUserTotalEarnings(address)._user (tokenVault.sol#1155) is not in mixedCase
Parameter tokenVault.multiCall(address,uint256)._user (tokenVault.sol#1168) is not in mixedCase
Parameter tokenVault.multiCall(address,uint256)._stakeID (tokenVault.sol#1168) is not in mixedCase
Parameter tokenVault.viewPoolPayout(address)._contract (tokenVault.sol#1181) is not in mixedCase
Parameter tokenVault.viewPoolMinServe(address)._contract (tokenVault.sol#1185) is not in mixedCase
Parameter tokenVault.getNrOfStakes(address)._user (tokenVault.sol#1192) is not in mixedCase
Parameter tokenVault.payFee(tokenVault.UserInfo,address)._userAddress (tokenVault.sol#1209) is not in mixedCase
Constant tokenVault.maxFee (tokenVault.sol#860) is not in UPPER_CASE_WITH_UNDERSCORES
Constant tokenVault.maxFundingFee (tokenVault.sol#861) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
tokenVault.payFee(tokenVault.UserInfo,address) (tokenVault.sol#1209-1238) uses literals with too many digits:
        - commission = (block.timestamp - _lastAction) / 3600 * user.amount * fundingRate / 1000000 (tokenVault.sol#1224)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
tokenVault.sol analyzed (8 contracts with 84 detectors), 67 result(s) found
```

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
High	High level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium level vulnerabilities are important to fix; however, they cannot lead to lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

Audit Findings

Critical:

TokenVault

```
poolPayout[].amount = 100;
poolPayout[].minServe = 864000;

poolPayout[].amount = 300;
poolPayout[].minServe = 2592000;

poolPayout[].amount = 500;
poolPayout[].minServe = 5184000;

poolPayout[].amount = 1000;
poolPayout[].minServe = 8640000;

poolPayout[].amount = 2500;
poolPayout[].minServe = 20736000;

poolPayout[].amount = 10000;
poolPayout[].minServe = 31536000;
```

Indexing can not be blank . It has some syntax issues and doesn't follow the correct pattern for defining or initializing values for such data structures.

You should use a data structure such as a struct and an array or mapping to store the values.

2.

```
*/
function harvest() public {
    IMasterChef(masterchef).updatePool(poolID);
    uint256 _currentCredit = IMasterChef(masterchef).credit(address(this));
    uint256 _accumulatedRewards = _currentCredit - lastCredit;
    lastCredit = _currentCredit;
    accDtxPerShare+= _accumulatedRewards * 1e12  / stakeToken.balanceOf(address(this));
}
```

Harvest() is set to public as any user can call this function and the “accDtxPerShare” keeps increasing.

3.

“updateFees()”,”updateTreasury()”,”withdrawStuckTokens()” can be called by anyone.

```
function updateFees() external {
    uint256 _depositFee = IGovernor(IMasterChef(masterchef).owner()).depositFee();
    uint256 _fundingRate = IGovernor(IMasterChef(masterchef).owner()).fundingRate();

    require(_depositFee <= maxFee, "out of limit");
    require(_fundingRate <= maxFundingFee, "out of limit");

    depositFee = _depositFee;

    if(_fundingRate != fundingRate) {
        fundingRate = _fundingRate;
        lastFundingChangeTimestamp = block.timestamp;
    }
}
```

```
305     function withdrawStuckTokens(address _tokenAddress) external {
306         require(_tokenAddress != address(token), "illegal token");
307         require(_tokenAddress != address(stakeToken), "illegal token");
308
309         IERC20(_tokenAddress).safeTransfer(treasuryWallet, IERC20(_tokenAddress).balanceOf(address(this)));
310     }
```

```
295     function updateTreasury() external {
296         treasury = IMasterChef(masterchef).feeAddress();
297         treasuryWallet = IGovernor(IMasterChef(masterchef).owner()).treasuryWallet();
298     }
```

PlsVault

4.

```
poolPayout[].amount = 100;
poolPayout[].minServe = 864000;

poolPayout[].amount = 300;
poolPayout[].minServe = 2592000;

poolPayout[].amount = 500;
poolPayout[].minServe = 5184000;

poolPayout[].amount = 1000;
poolPayout[].minServe = 8640000;

poolPayout[].amount = 2500;
poolPayout[].minServe = 20736000;

poolPayout[].amount = 10000;
poolPayout[].minServe = 31536000;
```

Indexing can not be blank . It has some syntax issues and doesn't follow the correct pattern for defining or initializing values for such data structures.

You should use a data structure such as a struct and an array or mapping to store the values.

5.

“updateFees()”,”updateTreasury()”,”withdrawStuckTokens()” can be called by anyone.

High:

No high severity vulnerabilities were found.

Medium:

No medium severity vulnerabilities were found.

Low:

No low severity vulnerabilities were found.

Very Low:

No very low severity vulnerabilities were found.

Conclusion

We were given a contract file and have used all possible tests based on the given object. We have some critical issues so it is not ready for mainnet deployment. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

The security state of the reviewed contract is "**Poorly**-**Secured**".

Note For Contract Users

There are several administrator functions. Those can be called by the administrator's wallet only. So, if the administrator's wallet is compromised, then it carries the risk of the contract becoming vulnerable.

SetMasterchefAddress: This function means to update the MasterChef contract address and the associated pool ID, and it enforces access control to ensure that only authorized entities can make these changes. This is a crucial feature for allowing the TokenVault contract to adapt to different MasterChef contracts or pools as needed during its operation.

UpdateSettings: The UpdateSettings function allows an external entity with the “decentralizedVoting” permission to update a setting in the contract called defaultDirectPayout

SetPoolPayout: The SetPoolPayout function allows an external entity with the "decentralizedVoting" permission to set or update the payout settings for a specific pool identified by its address.

administrator has full control over the smart contract. Thus, technical auditing does not guarantee the project's ethical side.

Dependent contracts such as MasterChef, are not within the scope of this audit.

Please do your due diligence before investing. Our audit report is never an investment advice.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyse the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

RD Auditors Disclaimer

The smart contracts given for audit have been analysed in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Because the total number of test cases are unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

Email: info@rdauditors.com

Website: www.rdauditors.com