QuillAudits

Audit Report
May, 2023

For

Table of Content

Table of Content

Executive Summary

Project Name	Polylastic
Overview	Polylastic DAO contains index functionality where stakers can stake and own index tokens. Index has PartnerProgram functionality to reward referrers. There can be two types of indexes admin index and community index. indexes are managed by DAO's governance functionanality. Users can create proposals where users need to vote with POLX vote token to approve proposals.
Timeline	February 20, 2023 to April 4, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze Polylastic codebase for quality, security, and correctness.
Github Link	*https://github.com/Polylastic-POLX/Smart-Contracts/tree/main*
Commit hash	c4449432c3cc6bdc703bf3f69b9d98e631eab90b
Fixed In	04cc6ed21d8d0d7e7df2bd0c61c9c248f1c05ba9

High Medium Low Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	2	4	3
Partially Resolved Issues	0	0	1	0
Resolved Issues	2	2	11	4

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - DAOAdmin

High Severity Issues

No issues found

Medium Severity Issues

A1. Review the logic for admin DAOs

Description

Whitelisted addresses may not have all the total supply of POLX/voteToken. In calculation _minimumQuorumPercent of totalSupply for dao of whitelisted addresses may need different value for _minimumQuorumPercent as compared to community dao, as whitelisted users would be holding specific percent of total supply.

There can be a case where whitelisted addresses can take tokens from non-whitelisted addresses and can use them to vote which can help them to approve the proposals.

Recommendation

The impact depends on factors like if whitelisted addresses can be trusted, _minimumQuorumPercent set at that moment.Our recommendation is to verify the logic and redesign the logic in case there's a possibility of above mentioned scenarios.

Status

Acknowledged

Low Severity Issues

No issues found

Informational Issues

No issues found

B. Contract - DAOCommunity

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

C. Contract - DAOStandart

High Severity Issues

C.1 Voting amplification attack

Description

Malicious user can perform a Voting amplification attack if he stakes, votes and withdraws on the proposal.endTimeOfVoting timestamp.

Exploit scenario:

1. Malicious users stake some amount on proposal.endTimeOfVoting timestamp.
2. User votes for that proposal, it will be allowed to vote as the if condition doesn't pass and it doesn't revert if someone stakes on last block as it checks “block.timestamp > proposal.endTimeOfVoting” then only it reverts with DAOImpossibleVote (as shown in below snippet) and doesnt revert if its equal.
3. Then if a malicious user withdraws with _withdraw() , it will allow it to withdraw as on L138 checks “if (user.unlockBalance > block.timestamp)” then only it reverts the transaction. In this case it won’t revert because the user is withdrawing when user.unlockBalance and block.timestamp are equal.
4. Once withdrawn, he then sends it to his other address.
5. He then stakes and votes with the same amount for the same proposal from that address in the same transaction and on the same block.timestamp with his other address.

Here users were able to use the same amount to vote on the same proposal with different addresses. Here the main attack is possible because if() block in the _vote() allows to vote on proposal.endTimeOfVoting.

```
209         function _vote(uint256 proposalId, bool supportAgainst) internal virtual {
210             require(
211                 _voteStatus[proposalId][msg.sender] != true,
212                 "DAO: you have already voted"
213             );
214
215             Proposal storage proposal = _proposals[proposalId];
216             User storage user = _users[msg.sender];
217
218             if (
219                 block.timestamp > proposal.endTimeOfVoting ||
220                 block.timestamp < proposal.startTime ||
221                 proposal.votingStatus != VotingStatus.CREATE
222             ) {
223                 revert DAOImpossibleVote();
224             }
225
```

Recommendation

Consider using “block.timestamp => proposal.endTimeOfVoting” so that if current block.timestamp would be equal to proposal.endTimeOfVoting and if the user tries to vote then it will revert.

Status

Resolved

C.2 Possible flashloan attack

Recommendation

Flashloan attack is possible on the proposal.endTimeOfVoting in the same way explained in C.1 (assuming lending pool is there Or they lend other tokens and swap them for POLX/vote token). Here an attacker can take a flashloan from the lending pool and can perform the same process of staking, voting, withdrawing and paying flashloan amount+fees back to the lending pool.

Depending on the quorum set and amount taken using flashloan any proposal can be approved.

This attack is possible because of the same reason as the C.1 issue. because if() block in the _vote() allows voting on proposal.endTimeOfVoting.

Recommendation

As suggested in C.1 consider using “block.timestamp => proposal.endTimeOfVoting” so that if current block.timestamp would be equal to proposal.endTimeOfVoting and if the user tries to vote then it will revert.

Status

Resolved

Medium Severity Issues

C.3 Use of “tokens with fees” can lead to unexpected outcomes.

Description

While depositing using _deposit() it is assumed that the amount that it will enter will be transferred using safeTransferFrom and the same amount is getting added to _users[msg.sender].balance on L127. If _voteToken deducts some amount of fees on every transfer then it can create an unexpected scenario where the amount that will get transferred won’t be the same as the amount passed to the function. In this type of scenario a user can transfer one amount lets say 100 tokens and while withdrawing he can withdraw 100 tokens even though the transferred amount was 100-(tax).

Recommendation

Check the amount of _voteToken that contract holds before and after transfer and then subtract the amount that was saved before from the current amount. Then this amount can be added to _users[msg.sender].balance.

Status

Resolved

C.4 inconsistency in use of PRECISION_E6

Description

PRECISION_E6 is getting used in denominator while calculating _minimumQuorumPercent in _finishVote() on L257. In _finishPoll() its not using PRECISION_E6 precision multiplier on L291 in denominator. If decimals are used then _minimumQuorumPercent should be according to PRECISION_E6.

Recommendation

Verify the logic, multiply PRECISION_E6 in denominator in _finishPoll() if it is getting used while assigning _minimumQuorumPercent , Otherwise remove the multiplication of PRECISION_E6 from denominator in _finishVote() on L257 if it is not getting used while assigning _minimumQuorumPercent.

Status

Resolved

Low Severity Issues

C.5 Add time delays

Description

Some time delays can be added so that after voting completes and _finishVote() executes there would be a waiting period to actually execute the proposed transaction so that in case any malicious proposal gets approved it can't be executed directly at that moment so that community will get some time to think and take actions against it.

Recommendation

We recommend to add a delay where team will get time to react. We recommend the project team to think on how much time would be enough to react and what can be done in that case depending on current logic implemented. e.g cancelling proposals or or pausing some functionalities on recipient address etc.

Status

Resolved

Informational Issues

No issues found

D. Contract - Index

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

D.1 Burning all amountLP and minting tax instead of transferring

Description

In _unstake() function while burning the indexLP it is burning the amountLP and its minting tax amount on L237. while comment says “// sending the tax to the treasure”.
Here eventually the same thing is happening. But the functionality can be confirmed once again.

Recommendation

Please confirm that it is intended functionality.

Status

Resolved

Auditor’s comment: changed the comment to to “mint the tax to the treasure”

D.2 Not getting exact amount back for the staked amount

Description

In stake() function there is call to _calcCost() function internally to get the price of LP token but what happens is that in _calcCost() when getPrice() is called as index contains 5 or more assets and each one of them having their own swap paths it is quite possible that 2 or more assets might have same token to go through so when this calls are made consecutively there can be a little difference in between which might result in not returning exact amount.

In the case where amountUSD += (amountUSD * 10) / PRECISION_E6 is used only to add more amount to the original amountUSD required to buy index assets, the other mechanism can be used where instead of transferring cost in stake() L192 , The “amountUSD parameter + slippage% of amountUSD” can be transferred so eventually it will fail if contract won't have enough value of _actualAcceptToken.

For stakeETH() there's no slippage parameter so in this case it can be handled on front-end level e.g while user would be sending transaction, the functionality can be allowed to the users where they can set the amount and the slippage (extra amount percentage they are willing to pay if cost exceeds from what was calculated)

In both of the scenarios dust amount (the amount that is not getting used to buy index assets) can be sent back to the user.

Recommendation

Consider changing logic as suggested in description.

Status

Acknowledged

Polylastic team’s comment: On the frontend side, we have already increased the "amauntUSD" by adding slippage.

D.3 Dust value after staking is not returned

Description

In stake() function the amount required to buy token can be less than the calculated cost amount required to buy intended amountLP. Again when all the swaps happen for the assets there can be difference in prices which keeping some dust amount of tokens in contract. This amount is not getting send back as mentioned in D.2.

Recommendation

Add a logic to send the dust assets back to users.

Status

Resolved

D.4 Rebalancing might fail with `UniswapV2Library: INSUFFICIENT_INPUT_AMOUNT`

There can be an extreme condition where there is a need to rebalance assets of a certain index because it might not be doing well as users are unstaking. So when rebalance is called on such an index considering _activeAssets[i].totalAmount is zero. When usdAmount is passed to _changeActualToken here it tries to exchange zero value resulting in `UniswapV2Library: INSUFFICIENT_INPUT_AMOUNT` error.

Recommendation

Always make sure that there is enough balance in the index when rebalance is called.

Status

Resolved

D.5 Frontrunning and Sandwich attack possibility

Description

There is possibility of front-running and sandwich attacks happening while staking, unstaking and rebalancing when trades are getting performed via quickswap when tokens get exchanged/swapped. While staking (using stake()) there would be limited amount of actual accept tokens would be present in the contract sent by the staker (i.e msg.sender) to buy index tokens. So even though frontrunning can happen for the trade(s) and even though tokens in the pool become expensive, because the contract wont have enough actual token amount to buy those expensive index token using swapTokensForExactTokens() , the transaction will revert.
If there would be the case where “actual accept token” and index asset would be same, then in this case because of frontrunning when index asset(s) becomes expensive, to fulfill the trade exchange can use the amount (apart from what staker sent) of index asset that contract would be holding (which was bought by contract to buy other user’s index token). Our recommendation would be to not have same actual accept token and index token to avoid this scenario to do this add a check which will revert for same token address.

Example (for rebalance and unstake scenario):
1) While rebalancing (and while unstaking) tokens, _swapDEX() is getting called which is using swapExactTokensForTokens() from router to swap amount.
2) attacker can observe and knows which assets are in index and then creates and executes a transaction with huge amount to swap on quickswap(exchange).
3) which changes the price for the index assets which might lead to getting the assets in much lower quantity.

Recommendation

1. While rebalancing check that the newAssets does not contain the current accept token by adding a require check which will check this in the loop for every index asset in newAssets.
2. For the remediation looking at design limitations add a solution where slippage can be specified and amount that contract will receive can be calculated and can be specified while calling the rebalance() and unstake() function so that it can be verified that contract is not receiving less value and if it receives the less value than specified slippage percentage the transaction can be reverted.

Status

Partially Resolved

Auditor’s comment: Setting status to partially resolved as no check added for remediation point 1. And slippage is not getting checked in _unstake() for minAmount so transaction can revert even if the sum amount would be fractionally less.

D.6 Centrailization in rebalancing process

Description

rebalance() can be called by only ADMIN_ROLE ,because of some design limitations mentioned in D.5 admin would be specifying calculatedPrice while calling the rebalance() so that it can be checked with the newPrice returned after rebalancing. A malicious admin can specify the slippage or expected amount in the way where it will let the contract accept less amount of tokens after trade.

Recommendation

A multisig wallet can be used which will allow to call rebalance() where centralization can be eliminated and calculatedPrice parameter can be specified after calculation.

Status

Acknowledged

Informational Issues

D.7 Do not use fee on transfer token

Description

This contracts stake and unstake functionality doesn't support for fee on transfer tokens. If fee on transfer tokens are used please make sure that how many tokens are getting sent to the contract.

Remediation

Do not use the fee on transfer tokens in assets of the index and as accept token.

Status

Acknowledged

D.8 Redundant variable calculation

Description

_amountTax variable in Index.sol contract is used for storing the whole tax amount but it is not used in any of the calculations.

Remediation

Can be removed if not used.

Auditor's comment: Added getter method for _amountTax.

Status

Resolved

D.9 Functions not used

Description

_exchangeDEX() and _buyNewAssets() functions have not been used or referenced anywhere in the code.

Remediation

Can be removed if not used.

Status

Resolved

E. Contract - IndexAdmin

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

F. Contract - IndexCommunity

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

G. Contract - PartnerProgram

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

G.1 A user can refer his own addresses

Description

The partnerProgram has a referrer system where users can refer other users to get on the program. But here the token amount is not getting transferred from the referred user to referrer but it gets minted so they can create many such accounts to mint rewards to themselves.

Recommendation

Verify the business logic , as highlighted in the description a user can create many addresses to get rewards minted to himself. If this breaks the logic then the logic needs to be changed. If the team is aware about this scenario and belive it doesnt break the business logic then the issue can be acknowledged.

Status

Acknowledged

G.2 General Recommendation

Description

treasure parameter in the constructor is not getting used and can be removed.

Recommendation

Remove the unusned constructor parameter.

Status

Resolved

Informational Issues

No issues found

H. Contract - IndexLP

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

H.1 Redundant minting

Description

Constructor mints 0 amount to the contract address which is redundant.

```
11          constructor(address partnerProgram) ERC20("Polylastic LP", "ILP") {
12              _setupRole(MINTER_ROLE, msg.sender);
13              _setupRole(BURNER_ROLE, msg.sender);
14              _setupRole(MINTER_ROLE, partnerProgram);
15              _mint(address(this), 0);
16          }
17
```

Recommendation

Check and remove the minting of 0 tokens to contract itself.

Status

Resolved

Informational Issues

No issues found

I. Contract - POLX (PolylasticV3)

High Severity Issues

No issues found

Medium Severity Issues

I.1 Fees on transferFrom can be bypassed.

Description

transfer() is overridden for deducting and burning fees, so when someone transfers an amount of token, some amount gets burned and some amount gets sent to the treasury according to burnPercentage and treasuryPercentage.
But users can use transferFrom to bypass deduction of fees as transferFrom() doesn't call transfer() , it calls _transfer internal function to perform transfer of tokens (similar to transfer()).

In the case where its need to perform fee deductions on transfer() as well as transferFrom(), _beforeTokenTransfer() hook can be overridden which is getting called in _transfer() and _transfer() is getting called in both transfer() and transferFrom().

Recommendation

Override _beforeTokenTransfer() hook instead of transfer() to deduct fees on transfer() as well as transferFrom(). *Reference*.

Status

Acknowledged

Polylastic team's comment: Currently for deployed POLX contract Polylastic team has set burnPercentage and treasuryPercentage as 0. So overall tax is set to 0.

Low Severity Issues

I.2 Events can be added for critical functions

POLX token contract contains some critical functions e.g updateBurnPercentage(), updateTreasuryPercentage(). Events can be added for these actions if necessary.

Recommendation

Add events for these actions

Status

Acknowledged

Informational Issues

I.3 unnecessary import statement

Description

There's unnecessary import for IERC20 as IERC20 is getting imported in ERC20.sol which is getting imported and inherited by PolylasticV3.

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity 0.8.10;
3
4   import { IERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5   import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
6   import { ERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
7   import { Ownable } from "@openzeppelin/contracts/access/Ownable.sol";
8   import "@openzeppelin/contracts/utils/Address.sol";
```

Remediation

Redundant IERC20 import can be removed.

Status

Acknowledged

I.4 Cannot set and find percentage less than 1

Description

For finding burnPercentage and the treasuryPercentage percentage of amount the denominator used (on L 59, L69) is 100. Using current logic, the smallest percentage that can be calculated is 1. If this is required to calculate a percentage in the form of decimals then there's need to add big amount on denominator e.g 1000
if we want to find 0.5% of 100e18 it is not possible in current logic , so if we add more zeros on the denominator side it can allow us to find this much percentage.
e.g to find 1 percent we specify burnPercentage or treasuryPercentage as 10 with denominator as 1000.
100e18 * 10 / 1000 = 10000000000000000 (1e18)
So to find 0.5 percent ,we will specify burnPercentage or treasuryPercentage as 5 with denominator as 1000.
100e18 * 5 /1000 = 5000000000000000 (0.5e18)

```
51          function transfer(address recipient, uint256 amount) public override returns (bool) {
52              require(recipient != address(0), "ERC20: transfer to the zero address");
53              if (
54                  _msgSender() != bridgeContract &&
55                  recipient != bridgeContract &&
56                  _msgSender() != treasury &&
57                  recipient != treasury
58              ) {
59                  uint256 burnableTokens = (amount * burnPercentage) / 100;
60                  uint256 treasuryTokens = (amount * treasuryPercentage) / 100;
61                  uint256 taxedTokens = burnableTokens + treasuryTokens;
62                  _burn(_msgSender(), burnableTokens);
63                  super.transfer(treasury, treasuryTokens);
64                  super.transfer(recipient, (amount - taxedTokens));
65                  return true;
66              }
67              super.transfer(recipient, amount);
68              return true;
69          }
```

Recommendation

If it is required to calculate percentage in decimals then more zeros can be added on the denominator side as described in description.

Status

Acknowledged

J. Contract - Treasure

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

K. Contract - FactoryAdmin

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

K.1 Redundant import

Description

There's redundant import for "hardhat/console.sol" on L9. This import should be removed before deploying contracts.

```
4   import "@openzeppelin/contracts/proxy/Clones.sol";
5   import "@openzeppelin/contracts/access/AccessControl.s
6   import "contracts/index/IIndex.sol";
7   import "../partnerProgram/IPartnerProgram.sol";
8   import "./IFactory.sol";
9   import "hardhat/console.sol";
10  import "../index/IIndexAdmin.sol";
```

Recommendation

Remove redundant import of "hardhat/console.sol".

Status

Resolved

Informational Issues

No issues found

L. Contract - FactoryCommunity

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

L.1 Incorrect event parameter passed

Description

changeIndexMaster() function changes the index master address and emits ChangeIndexMaster event. This event takes two parameters that are oldIndexMaster and newIndexMaster.

In the changeIndexMaster() function it sets newIndexMaster to _indexMaster , after that it emits ChangeIndexMaster event. But because newIndexMaster is set to _indexMaster , while emitting it emits the same address for oldIndexMaster and newIndexMaster.

Recommendation

Use a temporary variable to store oldIndexMaster so it can be used while event Or emit event before the assignment happens.

Status

Resolved

L.2 Incorrect event emission

Description

changeMainParam() sets both _DAOAdminAddress and _DAOCommAddress addresses. It emits changeMainParam() event , It takes the first parameter as DAOAddress. While passing this parameter in the changeMainParam() function it passes _DAOAdminAddress.

Recommendation

Check if the first argument passed in the event is intended or use a different event which will allow passing both DAOAdminAddress and DAOACommAddress which are getting set.

Status

Resolved

Informational Issues

No issues found

M. Contract - ExtensionPause

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

M.1 Comment mismatch

Description

Contract contains a comment which says “operation of functions using the "suspended" modifier is stopped”, The contract doesn’t contain any modifier named suspended.

Recommendation

Remove comment reference to suspended modifier as it is not present in the contract.

Status

Resolved

N. Contract - ExtensionReferralSystem

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

N.1 Redundant function

Description

ExtensionReferralSystem is getting inherited by the PartnerProgram contract. _setOwner() is not getting implemented in the PartnerProgram contract which makes this function redundant.
Owner address is getting used in _distributeTheReward while assigning address to newReferral on L59. currently becaues there’s no other methods that calls _setOwner() it can't be assigned.

Recommendation

Add method which will call this internal method so that _owner can be assigned.

Status

Resolved

Auditor’s comment: Polylastic team has removed _setOwner internal function and the use of _owner from the _distributeTheReward().

Informational Issues

No issues found

O. Contract - ExtensionSelector

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

P. Contract - ExtensionWhiteList

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

No issues found

Q. Common Issues

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

Q.1 Floating pragma

Description

Contracts are using floating pragma (pragma solidity ^0.8.10) Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Using floating pragma does not ensure that the contracts will be deployed with the same version. It is possible that the most recent compiler version gets selected while deploying a contract which has higher chances of having bugs in it.

Recommendation

Remove floating pragma and use a specific compiler version with which contracts have been tested.

Status

Resolved

Informational Issues

Q.2 Add events for critical actions

Description

_setPercentReward() in ExtensionReferralSystem contract should have event emitted.

Recommendation

Please add events for the above mentioned functions

Status

Resolved

Functional Testing

Some of the tests performed are mentioned below

DAOAdmin

- ✓ Check If Deposit And Withdraw Works Simultanously()
- ✓ Revert If Timestamp Is Not Correct()
- ✓ Revert If User Has Already Voted()
- ✓ Revert If Voting Has Not Started()
- ✓ Add Proposal()
- ✓ Check Getters()
- ✓ Check If User Can Withdraw After Vote()
- ✓ Check Deposit Function()
- ✓ Check Poll Voting()
- ✓ Check voting()

IndexLP

- ✓ Test Should Fail If Other Than Owner Tries To Mint()
- ✓ Test Should Fail If Partner Should Not Be Able To Burn Tokens()
- ✓ Test Burn()
- ✓ Test Check Getters()
- ✓ Check If Owner Can Burn Tokens Of Other User()
- ✓ Test Mint()

chefV2

- ✓ Test Fuzz The Tranfer Function(address,uint256)
- ✓ Check If Percentage Gets Deducted If TransferFrom Is Used()
- ✓ Set Bridge Contract Address()
- ✓ Update Burn Percentage()
- ✓ Update Treasury Percentage()
  Should deduct fee on transfer as well as for transferFrom

Index

- ✓ Should be able to stake,unstake when _feeStake is non zero
- ✓ Should be able stake, rebalance with new ActualToken and unstake staked amount
- ✓ Multiple users should be able to stake and unstake.
- ✓ Multiple users stake then rebalancing happens with different assets, users unstake, when non zero _feeStake
- ✓ Should be able stake, unstake half amount, rebalancing happens for different assets and users should be able stake again

Automated Testing

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Polylastic. We performed our audit according to the procedure described above.

Some issues of High,Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Polylastic Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Polylastic Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$16B
Secured

800K
Lines of Code Audited

Follow Our Journey

Audit Report
May, 2023

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com