Eigenpie

Security Audit Report

February 14, 2024

Contents

1 Introduction **3**
1.1 About Eigenpie 3
1.2 Source Code 3

2 Overall Assessment **4**

3 Vulnerability Summary **5**
3.1 Overview 5
3.2 Security Level Reference 6
3.3 Vulnerability Details 7

4 Appendix **13**
4.1 About AstraSec 13
4.2 Disclaimer 13
4.3 Contact 14

1 Introduction

1.1 About Eigenpie

Eigenpie is a re-staking platform for SubDAO, providing Liquid Stake Token (LST) holders with the ability to re-stake their assets and maximize their profit potential. It achieves this by creating dedicated liquidity restaking for each accepted LST on its platform, effectively isolating risks associated with any particular LST.

1.2 Source Code

The following source code was reviewed during the audit:

- https://github.com/magpiexyz/eigenpie.git
- Commit ID: 297d1ba

And this is the final version representing all fixes implemented for the issues identified in the audit:

- https://github.com/magpiexyz/eigenpie.git
- Commit ID: 72227d5

2 | Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the `Eigenpie` project. Throughout this audit, we identified a total of 5 issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	3	-	-	3
Medium	1	1	-	-
Low	1	-	-	1
Informational	-	-	-	-
Undetermined	-	-	-	-

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~H-1~~ Revised Pre-Deposit Logic in EigenpieStaking::depositAsset()

~~H-2~~ Improper exchangeRate Precision in PriceProvider::updateMLRTPrice(address)

~~H-3~~ Improper Implementation of PriceProvider::updateMLRTPrice(address, uint256)

M-1 Potential Risks Associated with Centralization

~~L-1~~ Integration of Non-Standard ERC20 Tokens

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Description
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

[H-1] Revised Pre-Deposit Logic in EigenpieStaking::depositAsset()

Target	Category	IMPACT	LIKELIHOOD	STATUS
EigenpieStaking.sol EigenpiePreDepositHelper.sol	Business Logic	High	High	Addressed

The `EigenpieStaking::depositAsset()` function serves as a mechanism for users to deposit supported `LST` (e.g., `ankrETH`, `cbETH`, etc.) and the corresponding `mLRT-LST` token is minted. During the pre-deposit phase of the protocol, users deposit underlying token into the `EigenpieStaking` contract (line 177). However, the corresponding `mLRT-LST` token is not immediately minted for them (lines 166 - 168). Instead, they need to wait until the current pre-deposit cycle is concluded. Upon claiming (line 84), the `mLRT-LST` token will then be minted and allocated to the users (line 93). This may result in the `totalSupply` of the `mLRT-LST` token is not updated in time, which is crucial for calculating the `mLRT-LST/LST` exchange rate. Consequently, it will lead to inaccuracy in the exchange rate calculation.

EigenpieStaking::depositAsset()

```
145 function depositAsset(
146     address asset,
147     uint256 depositAmount,
148     uint256 minRec,
149     address referral
150 )
151     external
152     whenNotPaused
153     nonReentrant
154     onlySupportedAsset(asset)
155 {
156     // checks
157     if (depositAmount == 0  depositAmount < minAmountToDeposit) {
158         revert InvalidAmountToDeposit();
159     }

161     if (depositAmount > getAssetCurrentLimit(asset)) {
162         revert MaximumDepositLimitReached();
163     }

165     uint256 mintedAmount;
166     if (isPreDeposit) {
167         (mintedAmount,) = getMLRTAmountToMint(asset, depositAmount);
168         IEigenpiePreDepositHelper(eigenpiePreDepositHelper).feedUserDeposit(msg.
                sender, asset, mintedAmount);
169     } else {
```

```
170             // mint receipt
171             mintedAmount = _mintMLRT(asset, depositAmount);
172         }
173         if (mintedAmount < minRec) {
174             revert MinimumAmountToReceiveNotMet();
175         }

177         IERC20(asset).safeTransferFrom(msg.sender, address(this), depositAmount);

179         emit AssetDeposit(msg.sender, asset, depositAmount, referral);
180     }
```

EigenpiePreDepositHelper::userClaim()

```
84  function userClaim(uint256[] calldata _cycles, address[] calldata _assets)
        external nonReentrant {
85      for (uint256 i = 0; i < _cycles.length; i++) {
86          if (!claimmableCycles[_cycles[i]]) revert ClaimCycleNotStarted();
87          for (uint256 j = 0; j < _assets.length; j++) {
88              bytes32 cycleUserKey = this._getCycleUserKey(_cycles[i], msg.sender)
                    ;
89              UserInfo storage user = userInfo[cycleUserKey][_assets[j]];
90              uint256 amount = user.amount - user.claimed;
91              if (amount > 0) {
92                  address receipt = eigenpieConfig.mLRTReceiptByAsset(_assets[j]);
93                  IMintableERC20(receipt).mint(msg.sender, amount);
94                  user.claimed += amount;
95                  emit Claim(msg.sender, _assets[j], amount, _cycles[i]);
96              }
97          }
98      }
99  }
```

Remediation Ensure the `totalSupply` of the `mLRT-LST` token is updated in time.

[H-2] Improper exchangeRate Precision in PriceProvider::updateMLRTPrice(address)

Target	Category	IMPACT	LIKELIHOOD	STATUS
PriceProvider.sol	Business Logic	High	High	Addressed

The `PriceProvider::updateMLRTPrice(address)` function is utilized to update the `mLRT-LST/LST` exchange rate for the specified asset. The exchange rate is derived from the current state of the corresponding pool. During our examination of the exchange rate calculation logic, it is apparent

that there is a loss of precision for the result. Given this, we suggest to improve its implementation as below: `uint256 exchangeRate = totalLST * 1 ether / receiptSupply` (line 69).

Moreover, to mitigate potential front-run attacks, we recommend adding access control to this function and execute transactions for updating the exchange rate through private RPC (e.g., `flashbot`).

PriceProvider::updateMLRTPrice(address)

```
54  /// @notice updates mLRT-LST/LST exchange rate
55  /// @dev calculates based on stakedAsset value received from eigen layer
56  /// @param asset the asset for which exchange rate to update
57  function updateMLRTPrice(address asset) external {
58      address mLRTReceipt = eigenpieConfig.mLRTReceiptByAsset(asset);
59      uint256 receiptSupply = IMLRT(mLRTReceipt).totalSupply();

61      if (receiptSupply == 0) {
62          IMLRT(mLRTReceipt).updateExchangeRateToLST(1 ether);
63          return;
64      }

66      address eigenStakingAddr = eigenpieConfig.getContract(EigenpieConstants.
            EIGENPIE_STAKING);
67      uint256 totalLST = IEigenpieStaking(eigenStakingAddr).getTotalAssetDeposits(
            asset);

69      uint256 exchangeRate = totalLST / receiptSupply;

71      _checkNewRate(mLRTReceipt, exchangeRate);

73      IMLRT(mLRTReceipt).updateExchangeRateToLST(exchangeRate);
74  }
```

Remediation Correct the implementation of the `PriceProvider::updateMLRTPrice(address)` function as above mentioned.

[H-3] Improper Implementation of PriceProvider::updateMLRTPrice(address, uint256)

Target	Category	IMPACT	LIKELIHOOD	STATUS
PriceProvider.sol	Business Logic	High	High	Addressed

As part of its intended functionality, the `PriceProvider::updateMLRTPrice(address, uint256)` function is employed by the privileged account to manually adjust the exchange rate based on off-chain calculations, thereby optimizing gas usage. However, thorough examination of its implementation,

we observed that it lacks any form of access control and does not actually modify the exchange rate, which clearly deviates from the intended design.

PriceProvider::updateMLRTPrice(address, uint256)

```
76  /// @notice updates mLRT-LST/LST exchange rate manually for gas fee saving
77  /// @dev calculates based on stakedAsset value received from eigen layer
78  /// @param asset the asset for which exchange rate to update
79  /// @param newExchangeRate the new exchange rate to update
80  function updateMLRTPrice(address asset, uint256 newExchangeRate) external {
81      address mLRTReceipt = eigenpieConfig.mLRTReceiptByAsset(asset);

83      _checkNewRate(mLRTReceipt, newExchangeRate);

85      emit ExchangeRateUpdate(asset, mLRTReceipt, newExchangeRate);
86  }
```

Remediation Apply necessary access control and properly update the exchange rate.

[M-1] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Medium	Medium	Acknowledged

In the `Eigenpie` protocol, the existence of a series of privileged accounts introduces centralization risks, as they hold significant control and authority over critical operations governing the protocol. In the following, we show the representative function potentially affected by the privileges associated with the privileged accounts.

MLRT::mint()/burnFrom()

```
67  /// @notice Mints EGETH when called by an authorized caller
68  /// @param to the account to mint to
69  /// @param amount the amount of EGETH to mint
70  function mint(address to, uint256 amount) external onlyRole(EigenpieConstants.
        MINTER_ROLE) whenNotPaused {
71      _mint(to, amount);
72  }

74  /// @notice Burns EGETH when called by an authorized caller
75  /// @param account the account to burn from
76  /// @param amount the amount of EGETH to burn
77  function burnFrom(address account, uint256 amount) external onlyRole(
        EigenpieConstants.BURNER_ROLE) whenNotPaused {
78      _burn(account, amount);
```

```
79 }
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged accounts. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

Response By Team This issue has been confirmed by the team. The multi-sig mechanism will be used to mitigate this issue.

[L-1] Integration of Non-Standard ERC20 Tokens

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Business Logic	Low	Low	Addressed

Inside the `EigenpieStaking::depositAsset()` function, the statement of `if (!IERC20(asset).transferFrom(msg.sender, address(this), depositAmount)) {revert TokenTransferFailed();}` (line 69) is employed to transfer the user's asset into the `EigenpieStaking` contract. However, in the case of `USDT-like` token whose transferFrom() lacks a return value, it would lead to a revert. Given this, we recommend employing the widely-used `SafeERC20` library (which serves as a wrapper for ERC20 operations while accommodating a diverse range of non-standard ERC20 tokens) to address this case.

EigenpieStaking::depositAsset()

```
128 function depositAsset(
129     address asset,
130     uint256 depositAmount,
131     uint256 minRec,
132     address referral
133 )
134     external
135     whenNotPaused
136     nonReentrant
137     onlySupportedAsset(asset)
138 {
139     // checks
140     if (depositAmount == 0  depositAmount < minAmountToDeposit) {
141         revert InvalidAmountToDeposit();
142     }

144     if (depositAmount > getAssetCurrentLimit(asset)) {
145         revert MaximumDepositLimitReached();
146     }
```

```
148     if (!IERC20(asset).transferFrom(msg.sender, address(this), depositAmount)) {
149         revert TokenTransferFailed();
150     }

152     // mint receipt
153     uint256 mintedAmount = _mintMLRT(asset, depositAmount);
154     if (mintedAmount < minRec) {
155         revert MinimumAmountToReceiveNotMet();
156     }

158     emit AssetDeposit(msg.sender, asset, depositAmount, referral);
159 }
```

Remediation Replace `transfer()/transferFrom()` with `safeTransfer()/safeTransferFrom()`.

4 Appendix

4.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

4.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

4.3 Contact

Name	AstraSec Team
Phone	+86 176 2267 4194
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI