Eigenpie

Security Audit Report

January 27, 2024

Contents

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

The SHA256 Hash of each audited file is as follows:

Table 1.1: SHA256 of Audited Files

Contract	SHA256
EigenpieConfig.sol	ce597989487e85167b16638e447f5c09e2b49107259df8358b9682e368b077ec
EigenpieStaking.sol	e04e5cd6f4e6073090168ed21b4285847a338c83b0078dffcd321c0c16453f9c
NodeDelegator.sol	5803e28d109d09a1e44f5311933e452a635cb521c411acee8d15905c5cbf50e6
MLRT.sol	28a386f5d6602d2dcb228466fb4d1c60b08188eee92939fdada01d7bf322bb5e
EigenpieConfigRoleChecker.sol	a137221d7a6318bc392775e8dab3d78c98392e798e44cc979c66827b78ee735e
EigenpieConstants.sol	6f900950c86e7685d17ba7bd55a10849943a97e1abb3eb86e4bcbc138d74fa46
UtilLib.sol	9931a0c168df87b2cd0a4b86d67b29a8657d0db1f385481d6eb520f49f5c23b7
PriceProvider.sol	d247275b6ee1d60aa9309c770a4737052cfcbbc4899544a099fed318cc60ed8e

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 4 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	2	-	-	2
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.

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] 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)` func-

tion as above mentioned.

[H-2] 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