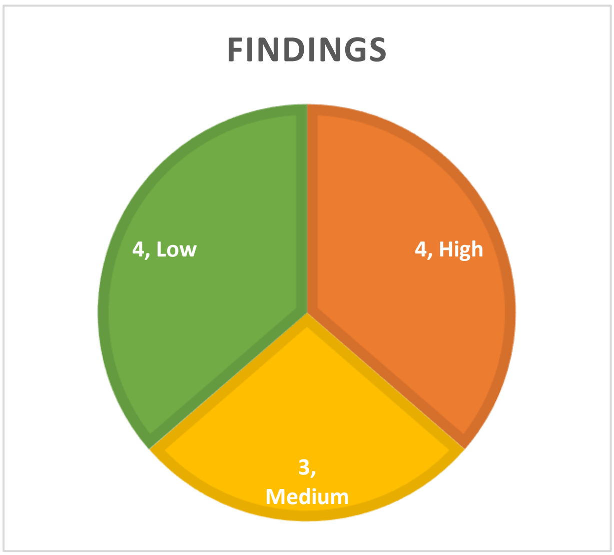
Trust Security

Smart Contract Audit

Mozaic Archimedes upgrade

08/08/2023

Executive summary

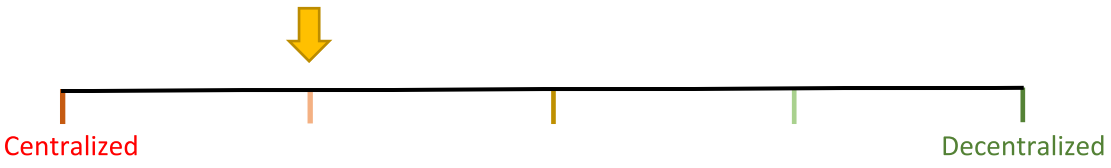


Category	Yield Aggregator
Audited file count	5
Lines of Code	1227
Auditor	HE1M
Time period	01/08-07/08

Findings

Severity	Total	Open	Fixed	Acknowledged	New issues
High	4	-	3	1	1
Medium	3	-	3	-	-
Low	4	-	3	1	-

Centralization score



Signature

Digitally signed by Trust
Date: 2023-08-14 12:47:23

Document properties

Versioning

Version	Date	Description
0.1	08/08/2023	Client report
0.2	14/08/2023	Mitigation review

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

Scope

- contracts/Controller.sol
- contracts/MozBridge.sol
- contracts/MozaicLp.sol
- contracts/StargatePlugin.sol
- contracts/Vault.sol

Repository details

- **Repository URL:** https://github.com/Mozaic-fi/Mozaic-Vault
- **Commit hash:** 2c6260a18f476394b6d79c8b4e184534ffc97db0
- **Mitigation commit hash:** 5e384a4a84c41c46e58507d5c22419630198dbb7

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Trust is the leading auditor at competitive auditing service Code4rena, reported several critical issues to Immunefi bug bounty platform and is currently a Code4rena judge.

About the Auditors

HE1M has achieved multiple top 5 in Code4rena contests and was active in more than 30 public bug-hunting events. He especially likes Layer 2s and bridging cross-chain projects. He has recently achieved 2nd place in Optimism's $700k contest. His unique line of thought leads him to uncovering extraordinary findings missed by all others in several contests.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Qualitative analysis

Metric	Rating	Comments
Code complexity	**Good**	Project kept code as simple as possible, reducing attack risks
Documentation	**Moderate**	Project is not very well documented.
Best practices	**Good**	Project is aware of industry standards and mostly follows them well.
Centralization risks	**Moderate**	In its current iteration, Mozaic holds significant power in management of user funds.

Findings

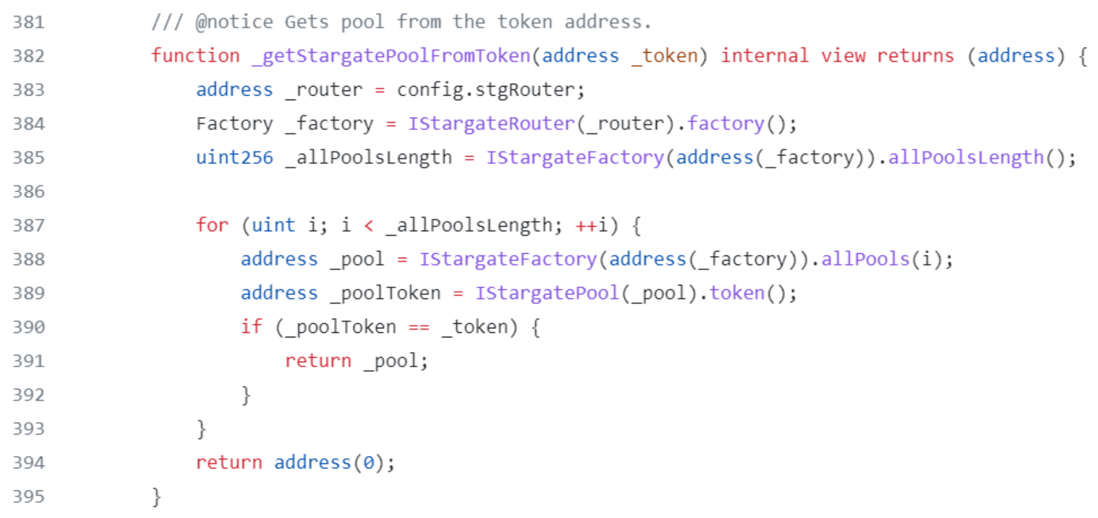
High severity findings

TRST-H-1 There is no way to withdraw the stargateToken from the Vault

- **Category:** Logical flaws
- **Source:** Vault.sol
- **Status:** Fixed + new issue

Description

There is no mechanism to withdraw the **_stargateToken** which is transferred to the vault during distributing rewards. One way was to swap this token to a stable token, but the pools of Stargate do not support this token yet. So, during swapping this token, no pool can be found, then the return address will be *address(0)*, and it will revert with error message "StargatePlugin: Invalid source token."



```
    /// @notice Gets pool from the token address.
    function _getStargatePoolFromToken(address _token) internal view returns (address) {
        address _router = config.stgRouter;
        Factory _factory = IStargateRouter(_router).factory();
        uint256 _allPoolsLength = IStargateFactory(address(_factory)).allPoolsLength();

        for (uint i; i < _allPoolsLength; ++i) {
            address _pool = IStargateFactory(address(_factory)).allPools(i);
            address _poolToken = IStargatePool(_pool).token();
            if (_poolToken == _token) {
                return _pool;
            }
        }
        return address(0);
    }
```

Recommended mitigation

A withdraw mechanism for reward token is required to be implemented.

Team response

Fixed.

Mitigation review

The function *bridgeViaLifi()* is added so that Master can easily swap it via LiFi.

```
    function bridgeViaLifi(
        address _srcToken,
        uint256 _amount,
        uint256 _value,
        bytes calldata _data
    ) external onlyMaster {
        require(
            address(LIFI_CONTRACT) != address(0),
            "Lifi: zero address"
        );
        bool isNative = (_srcToken == address(0));
        if (!isNative) {
            IERC20(_srcToken).safeApprove(address(LIFI_CONTRACT), 0);
            IERC20(_srcToken).safeApprove(address(LIFI_CONTRACT), _amount);
        }
        (bool success,) = LIFI_CONTRACT.call{value: _value}(_data);
        require(success, "Lifi: call failed");
    }
```

There is a significant centralization risk because the parameters **_srcToken** and **_data** can be arbitrarily inserted by the Master. For example, Master can swap/bridge the stablecoin deposited by the users to any address. However, it should be limited only to swap/bridge **_stargateToken** to the Vault.

TRST-H-2 An attacker can steal rewards during reporting of settlement

- **Category:** MEV attacks
- **Source:** Vault.sol
- **Status:** Fixed

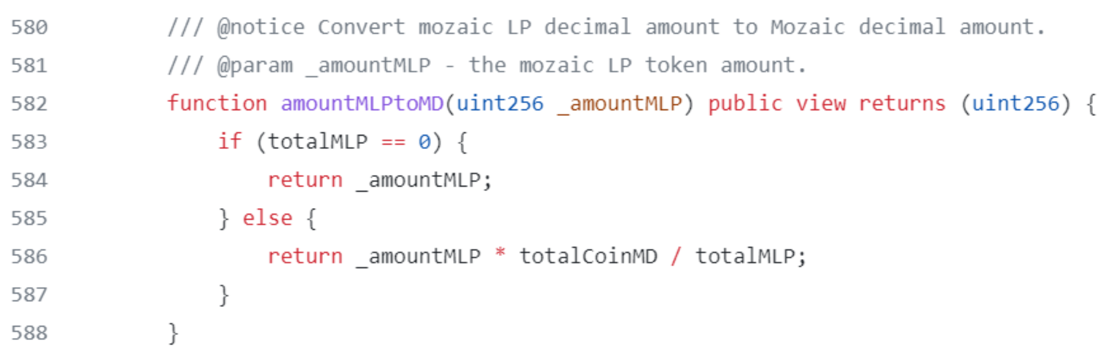
Description

In the settlement code flow, the Controller sends a TYPE_REQUEST_SETTLE to a destination chain. During reporting of "settled" in the destination chain, suppose an attacker notices that if this settlement is done, the ratio **totalMLP / totalCoinMD** on this chain will decrease. Let's say, this ratio is equal to **A**.

```
    /// @notice Convert Mozaic decimal amount to mozaic LP decimal amount.
    /// @param _amountMD - the token amount represented with mozaic decimal.
    function amountMDtoMLP(uint256 _amountMD) public view returns (uint256) {
        if (totalCoinMD == 0) {
            return _amountMD;
        } else {
            return _amountMD * totalMLP / totalCoinMD;
        }
    }
```

The attacker can perform a frontrunning attack. They deposit into the vault, and receive equivalent **mozLP** based on the ratio **A**. After the settlement, the ratio changes to **B**, where **B**

< A. Then, the attacker withdraws the deposited stablecoin by calling *addWithdrawRequest()*, receiving equivalent stablecoin based on the ratio **B**.



```
        /// @notice Convert mozaic LP decimal amount to Mozaic decimal amount.
        /// @param _amountMLP - the mozaic LP token amount.
        function amountMLPtoMD(uint256 _amountMLP) public view returns (uint256) {
            if (totalMLP == 0) {
                return _amountMLP;
            } else {
                return _amountMLP * totalCoinMD / totalMLP;
            }
        }
```

Since B < A, the attacker can gain profit by stealing the reward which is supposed to be other users' right who have their fund deposited in the Vault before.

The attack requires no preconditions and the funds lost depends on the attacker's resources and the pool amounts.

A simple example to show the concept:

Let's say in the destination chain, 5000 stablecoin is deposited and 5000 MozLP is minted. So, the ratio is **totalMLP/totalCoinMD = 1**. The attacker notices that the upcoming settlement will change the ratio to **totalMLP/totalCoinMD = 0.8**. So, the attacker deposits 20,000 stablecoin before the settlement and receives 20,000 MozLP. Now the state of the Vault is: **stablecoin = 25000, MozLP= 25000**. Then, after the settlement, the attacker burns 20,000 MozLP and receives *20,000/0.8 = 25,000*. So, the vault state will be **stablecoin = 0, MozLP= 5000**. The attacker gains 5000 stablecoin without keeping his fund there for a long time. Those users who own those 5000 mozLP, cannot withdraw until the next rebalancing, and they lost their reward.

Recommended mitigation

It is recommended to implement a fine for early withdrawal.

Team response

Fixed.

Mitigation review

A vault lock mechanism is implemented to lock the vault between the snapshot and settlement.

TRST-H-3 Draining the vault and stealing the rewards in case of reverted *reportSettled* by using flashloan

- **Category:** Economic attacks
- **Source:** Vault.sol
- **Status:** Fixed

Description

Suppose for any reason, reporting "settled" is reverted (this can be possible if the LayerZero relayer does not provide enough gas, or the Vault does not have enough native tokens to send back a message) in the MozBridge before the try/catch statement in the Vault. So, this transaction will be reverted, and get caught by the LayerZero Endpoint. Now, anyone can retry this transaction by calling *retryPayload()*.
An attacker notices that if this settlement is done, the ratio **totalMLP/totalCoinMD** on this chain will decrease. Let's say, this ratio before the settlement is equal to **A**.

```
    /// @notice Convert Mozaic decimal amount to mozaic LP decimal amount.
    /// @param _amountMD - the token amount represented with mozaic decimal.
    function amountMDtoMLP(uint256 _amountMD) public view returns (uint256) {
        if (totalCoinMD == 0) {
            return _amountMD;
        } else {
            return _amountMD * totalMLP / totalCoinMD;
        }
    }
```

So, the attacker gets a flashloan of a stablecoin, deposits into the vault, and receives equivalent **mozLP** based on the ratio **A**. In the same transaction, the attacker calls *retryPayload()* to retry the already failed report settled transaction. When this transaction is done, the ratio changes to **B**, where **B < A**. Then, in the same transaction, the attacker withdraws their deposited stablecoin by calling *addWithdrawRequest()*, and they receive equivalent stablecoin based on the ratio **B**.

```
    /// @notice Convert mozaic LP decimal amount to Mozaic decimal amount.
    /// @param _amountMLP - the mozaic LP token amount.
    function amountMLPtoMD(uint256 _amountMLP) public view returns (uint256) {
        if (totalMLP == 0) {
            return _amountMLP;
        } else {
            return _amountMLP * totalCoinMD / totalMLP;
        }
    }
```

Since B < A, the stablecoin received by the attacker will be higher than what they deposited first, so they can gain much profit by draining the vault and stealing the reward which is supposed to be other users' right who have their fund deposited in the Vault before.

Recommended mitigation

It is recommended to implement a mandatory time delay between the deposit and withdraw in the vault to protect against the flashloan possibility.

Team response

Fixed.

Mitigation review

A vault lock mechanism is implemented to lock the vault between the snapshot and settlement.

TRST-H-4 An attacker can drain Mozaic Vaults by manipulating the LP price

- **Category:** Economic attacks
- **Source:** Vault.sol
- **Status:** Acknowledged + *require attention*

Description

The controller is tasked with synchronizing LP token price across all chains. It implements a lifecycle. An admin initiates the snapshot phase, where Controller requests all Vaults to report the total stable ($) value and LP token supply. Once all reports are in, admin calls the settle function which dispatches the aggregated value and supply to all vaults. At this point, vaults process all deposits and withdrawals requested up to the last snapshot, using the universal value/supply ratio.

The described pipeline falls victim to an economic attack, stemming from the fact that LP tokens are LayerZero OFT tokens which can be bridged. An attacker can use this property to bypass counting of their LP tokens across all chains. When the controller would receive a report with correct stable value and artificially low LP supply, it would cause queued LP withdrawals to receive incorrectly high dollar value.

To make vaults miscalculate, attacker can wait for Controller to initiate snapshotting. At that moment, they can start bridging a large amount of tokens. They may specify custom LayerZero adapter params to pay a miniscule gas fee, which will guarantee that the bridge-in transaction will fail due to out-of-gas. At this point, they simply wait until all chains have been snapshotted, and then finish bridging-in with a valid gas amount. Finally, Controller will order vaults to settle, at which point the attacker converts their LP tokens at an artificially high price.

Another clever way to exploit this flaw is to count LP tokens multiple times, by quickly transporting them to additional chains just before those chains are snapshotted. This way, the LP tokens would be diluted and the attacker can get a disproportionate amount of LP tokens for their stables.

The dev team confirmed bridging would be configured but paused by the backend during snapshotting. Since such configuration flows are out of scope, the attack remains a high-severity finding in this iteration of the codebase.

Recommended mitigation

The easy but limiting solution is to reduce complexity and disable LP token bridging across networks. The other option is to increase complexity and track incoming/outgoing bridge requests in the LP token contract. When snapshotting, cross reference the requested snapshot time with the bridging history.

Team response

The team will disable LP bridging before snapshotting will take place.

Mitigation review

The mitigation chosen by the team is by off-chain prevention. Since configuration procedures are out of scope, users **should be aware** of the risk and trust assumptions.

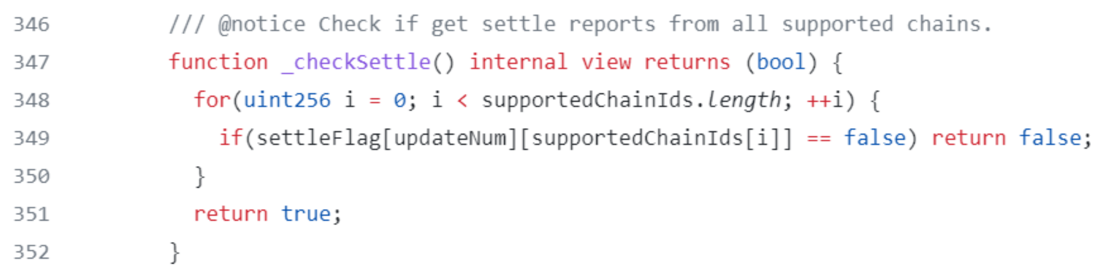
Medium severity findings

TRST-M-1 Protocol stuck if one chain is down

- **Category:** Logical flaws
- **Source:** Controller.sol
- **Status:** Fixed

Description

Suppose there are 5 supported chains by the protocol. The Master requests all the chains to update their status by calling *updateAssetState()*. So, the protocol status goes to SNAPSHOTTING mode. When the last remaining chain updates its status, the protocol status goes to OPTIMIZING mode if *_checkSettle()* returns true.



```
    /// @notice Check if get settle reports from all supported chains.
    function _checkSettle() internal view returns (bool) {
      for(uint256 i = 0; i < supportedChainIds.length; ++i) {
        if(settleFlag[updateNum][supportedChainIds[i]] == false) return false;
      }
      return true;
    }
```

Suppose, 4 chains have already updated their status, and the protocol is waiting the 5th chain to update its status, but this chain is down for any reason and is not responsive. In this case, the protocol cannot change the status from OPTIMIZING mode because it can be changed only if all the supported chains report their status.

Even if the 5th chain is removed by the Owner, it cannot solve the issue, because 4 chains already updated their status.

Note that requesting one of the already-reported chain to update its status again in order to trigger *_checkSettle(),* is not effective because it is already reported its status and its **settleFlag** is true.

This situation is not rare, as any chain can be temporarily down for some time, that can impact the protocol when it is rebalancing.

Recommended mitigation

There should be a mechanism to force changing the status of the protocol.

Team response

Fixed.

Mitigation review

The function *checkProtocolStatus()* is added to check the protocol status and change it if necessary.

TRST-M-2 Missing withdraw function in case of fee refund

- **Category:** Leak of value issues
- **Source:** Vault.sol
- **Status:** Fixed

Description

The Vault sets the LayerZero fee refund address to itself, but there is no mechanism to withdraw this amount in case of any refund. Additionally, the Vault accepts ETH transfers through fallback functions.

Recommended mitigation

Add a withdraw function in the Vault, which would move funds to the treasury.

Team response

Fixed

Mitigation review

The function *withdraw()* is added.

TRST-M-3 Removing a chain during SNAPSHOTTING

- **Category:** Logical flaws
- **Source:** Controller.sol
- **Status:** Fixed

Description

There is no required protocol status condition for removing a chain by the Owner. Let's say there are 5 supported chains by the protocol. Two possible scenarios:

1. During snapshotting, 5th chain is removed by the Owner. So, when 4th chain updates its status, the protocol status goes from SNAPSHOTTING to OPTIMIZING, and after settling all the 4 chains, the protocol status goes from OPTIMIZING to IDLE. Now, suppose, the 5th chain status is reported to the Controller by some delay. Since, it does not check whether this chain is a supported chain anymore or not, it will change the protocol status to OPTIMIZING again, because *_checkSnapshot* returns true (as **snapshotFlag** for all other chains are true).
2. During snapshotting, 5th chain is removed by the Owner. So, during settling, the status of 5th will not be included in **totalCoinMD** and **totalMLP** that can lead to miscalculation.

Recommended mitigation

The *removeChainId()* function should only be allowed during IDLE status.

A supported chain id check should be added to *updateSnapshot()*. Skip execution if the chain is not supported.

Team response

Fixed

Mitigation review

The supported chain id check is added to both functions *updateSnapshot()* and *settleReport()*.

Low severity findings

TRST-L-1 Unexpected behavior during settlement

- **Category:** Economical flaws
- **Source:** Vault.sol
- **Status:** Fixed

Description

When the protocol status is SETTLING, the new users who deposit into the Vault, may be exposes to a sudden change in the ratio of **totalMLP/totalCoinMD** after the settlement, that can impact the users financially.

Recommended mitigation

To prevent any unexpected behavior for users, it is recommended to stop any deposit into the Vault during non-IDLE status of the protocol, like what is implemented in LIDO that stops any transaction between 12:00 AM and 01:00 AM.

Team response

Fixed.

Mitigation review

The vault is locked between the snapshot request to settling request.

TRST-L-2 Missing try-catch during reportSettled()

- **Category:** Logical flaws
- **Source:** Controller.sol
- **Status:** Acknowledged

Description

The gas provided by the relayer in LayerZero is **_gasLimit**. Ignoring the overhead gas (for simplicity), (63/64)***_gasLimit** will be forwarded to Vault to *reportSettled()*. Again ignoring the overhead gas, (63/64)*(63/64)***_gasLimit** will be forwarded to MozBridge for the call back. If this amount of gas is not enough, then it will be reverted and caught by the catch-statement. So, the amount of gas available in the catch-statement will be (1/64) *(63/64) ***_gasLimit**. If this amount of gas is lower than the amount of gas needed for a SSTORE (20k), then the storing in the **revertLookup** will revert. This will be caught by the try-catch statement in LayerZero Endpoint. This will lock any next incoming messages to the protocol, until it is retried/unblocked by Vault, bypassing the important revert mitigations.

Recommended mitigation

The same try-catch used during requesting snapshot should be used for requesting settlement.

Team response

Acknowledged.

TRST-L-3 Not calculating the _assetsLD during snapshotting

- **Category:** Logical flaws
- **Source:** StargatePlugin.sol
- **Status:** Fixed

Description

When taking a snapshot, the total assets in the plugins should be taken into account as well. If the token balance of Plugin is nonzero, it will be transferred to the **localVault** and its amount will be added during calculating **_totalAssetsMD**. But if for any reason, the equivalent pool is not available for such token or the equivalent pool's index is not found, the for-loop will continue and the **_assetsLD** will not be added during calculation. This will cause some miscalculation in taking snapshot, because the amount of **_assetsLD** is not considered as the balance of the Plugin.

Recommended mitigation

The following modification is recommended for L300:

```
if(_pool == address(0)){
  if(_assetLD > 0){
    uint256 _assetsMD = convertLDtoMD(_token, _assetsLD);
    _totalAssetsMD = _totalAssetsMD + _assetsMD;
  }
  continue;
}
```

The following modification is recommended for L304:

```
if(found == false){
  if(_assetLD > 0){
    uint256 _assetsMD = convertLDtoMD(_token, _assetsLD);
    _totalAssetsMD = _totalAssetsMD + _assetsMD;
  }
  continue;
}
```

Team response

Fixed

Mitigation review

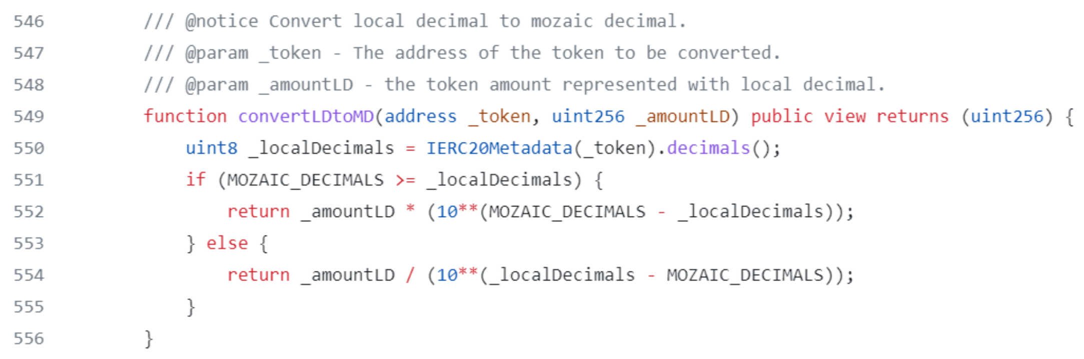
The recommended mitigation is added to the code.

TRST-L-4 Rounding errors may cause Vault to consume user's funds without granting any LP tokens.

- **Category:** Precision loss issues
- **Source:** Vault.sol
- **Status:** Fixed

Description

Due to mismatch between decimals, it is possible for loss of funds to occur during decimal conversions. For example, during depositing into the Vault, if **_localDecimals = 10** and **_amountLD = 1000** then **_amountLD / (10**(_localDecimals - MOZAIC_DECIMALS)) = 0** as **MOZAIC_DECIMALS = 6.**



```
    /// @notice Convert local decimal to mozaic decimal.
    /// @param _token - The address of the token to be converted.
    /// @param _amountLD - the token amount represented with local decimal.
    function convertLDtoMD(address _token, uint256 _amountLD) public view returns (uint256) {
        uint8 _localDecimals = IERC20Metadata(_token).decimals();
        if (MOZAIC_DECIMALS >= _localDecimals) {
            return _amountLD * (10**(MOZAIC_DECIMALS - _localDecimals));
        } else {
            return _amountLD / (10**(_localDecimals - MOZAIC_DECIMALS));
        }
    }
```

Recommended mitigation

Add a require condition to revert any deposit that leads to *_amountMLPToMint()* equal to zero.

Team response

Fixed.

Mitigation review

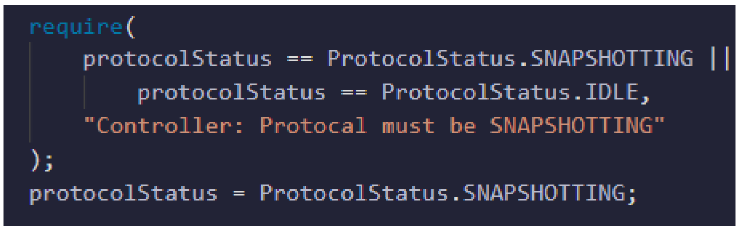
The recommended mitigation is added to the code.

Additional recommendations

TRST-R-1 Request for snapshot/settlement in a certain local vault

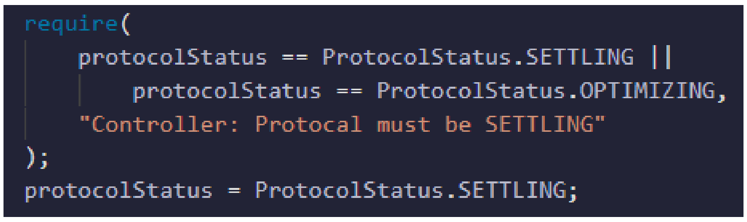
If Master would like to request for settlement or request for snapshot in a certain vault, it is not possible until once all the vaults are requested for settlement or snapshot. Because, to do so, the status should be SNAPSHOTTING or SETTLING. To give more flexibility to the Master, relaxing the conditions to the following can help:

At L224:



```
require(
    protocolStatus == ProtocolStatus.SNAPSHOTTING ||
        protocolStatus == ProtocolStatus.IDLE,
    "Controller: Protocal must be SNAPSHOTTING"
);
protocolStatus = ProtocolStatus.SNAPSHOTTING;
```

At L239:



```
require(
    protocolStatus == ProtocolStatus.SETTLING ||
        protocolStatus == ProtocolStatus.OPTIMIZING,
    "Controller: Protocal must be SETTLING"
);
protocolStatus = ProtocolStatus.SETTLING;
```

TRST-R-2 Cleaner usage of code

Lines 435 and 440 can be removed and placed after line 441.

The parameter **_refundAddress** is already defined as payable. So, no need to cast payable again. This occurs in several parts of codebase, like:

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/MozBridge.sol#L213

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/MozBridge.sol#L219

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/MozBridge.sol#L348

TRST-R-3 Tracking the constants

It is better to have a config contract to store all the settings to avoid any mismatch between the variables in two different contracts MozBridge and Vault.

TRST-R-4 Cache invariants

In some parts of the codebase, the gas usage can be improved by caching the storage variable.

bridgeLookup[_dstChainId]:

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/MozBridge.sol#L342

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/MozBridge.sol#L346

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/MozBridge.sol#L352

supportedPlugins[_pluginId]:

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/Vault.sol#L249-L265

snapshot:

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/Vault.sol#L475-L479

protocolStatus:

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/Controller.sol#L182C17-L187

totalCoinMD and **totalMLP**:

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/Controller.sol#L326-L335

TRST-R-5 Adhering to naming standards

Consider changing the name **totalCoinMD** and **totalMLP** to **_totalCoinMD** and **_totalMLP**, respectively in L235 and L262. Following naming standards reduces risks of confusion between state variables and input parameters.

TRST-R-6 Redundant zero approve

During staking, it is redundant to approve zero for LPToken in Stargate. The approve pattern should only be used for arbitrary tokens, which may revert when calling *approve()* when current approval is not zero.

TRST-R-7 Issues with comments

The comment is misleading as **msg.sender** is transferring, not the depositor.

The comment is misleading as it is the address of **_insurance**.

TRST-R-8 Typos/Grammars

There are some typos/grammars in the codebase, like:

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/MozBridge.sol#L259C56-L259C66

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/Vault.sol#L16C47-L16C54

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/Vault.sol#L71C53-L71C60

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/Vault.sol#L74C37-L74C47

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/Vault.sol#L167C56-L167C68

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/Vault.sol#L175C70-L175C82

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/Vault.sol#L183-L184

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/Controller.sol#L114C55-L114C67

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/Vault.sol#L191C76-L191C81 https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/Vault.sol#L192C104-L192C109

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/Vault.sol#L224C42-L224C47

https://github.com/Mozaic-fi/Mozaic-Vault/blob/2c6260a18f476394b6d79c8b4e184534ffc97db0/contracts/MozBridge.sol#L262C31-L262C33

Centralization risks

TRST-CR-1 Admin can change the LayerZero configuration

An admin can set up a custom relayer, which has significant power on all supported chains. For example, they can block execution of unwanted transactions.

TRST-CR-2 Admin can remove supported chainIds or plugins

Admins can perform economic attacks on the LP tokens by removing certain chainIds or plugins from calculations.

TRST-CR-3 Admin can drain all funds from the vault through malicious plugins

Admin can introduce malicious plugins and approve all vault assets through the *execute()* call. This represents a significant rug pull risk, in the event an admin key gets compromised.

TRST-CR-4 Owner can set treasury fee to 100%

Owner can set fee to 100% by calling *setFee* to transfer all the rewards to **localTreasury** or **localInsurance**.

Systemic risks

TRST-SR-1 Delivery of messages across chains

LayerZero is used to transport messages cross-chain. If a compromise of LayerZero take place, the worst case scenario may include forgeries of Mozaic messages, causing havoc and possibly loss of funds.

TRST-SR-2 Plugins

Any funds held by external plugins are exposed to third-party risks. Currently, that comprises of Stargate centralization-related threats and security compromise potential.