1. Introduction ................................................................................................................................................... 7
  1.1. Disclaimer................................................................................................................................................. 7
  1.2. Security Assessment Methodology .................................................................................................... 7
    1.2.1 Severity Level Reference ................................................................................................................ 8
    1.2.2 Status Level Reference.................................................................................................................... 8
  1.3. Project overview ..................................................................................................................................... 8
  1.4. Audit Scope.............................................................................................................................................. 8
  1.5. Steps to upgrade Lido on Polygon...................................................................................................... 9
    1.5.1 Contracts ........................................................................................................................................... 9
    1.5.2 Configuration .................................................................................................................................... 9
2. Report ............................................................................................................................................................ 10
  2.1. CRITICAL ................................................................................................................................................ 10
  2.2. MAJOR ................................................................................................................................................... 10
  2.3. WARNING.............................................................................................................................................. 10
    2.3.1 Incorrect totalStaked calculation................................................................................................ 10
    2.3.2 Rounding error in totalValidatorToWithdrawFrom calculation ............................................ 12
    2.3.3 Reentrancy in addNodeOperator function .............................................................................. 13
    2.3.4 DAO’s admin role not changing .................................................................................................. 13
    2.3.5 reservedFunds may become greater than expected .............................................................. 14
    2.3.6 Possible erroneous conversion rate from Matic to StMatic................................................. 15
    2.3.7 Possible erroneous conversion rate from StMatic to Matic................................................... 15
    2.3.8 No check for creating withdraw request with a zero shares which leads to DOS........ 16
    2.3.9 Withdraw requests that are never used and consume gas .................................................. 17
    2.3.10 Revert of _requestWithdrawBalanced in case of inactive operators .............................. 17
    2.3.11 Impossible to rebalance system if there are pending buffered tokens............................ 18
    2.3.12 Possible burning Matic without minting shares during delegate() ................................. 20
    2.3.13 Attackers can make calculatePendingBufferedTokens() fail with out-of-gas .......... 20
    2.3.14 Rounding error in amountToWithdraw calculation for operators with a small stake ... 21
2.4. INFO

2.4.1 Not descriptive variable name DISTANCE_THRESHOLD

2.4.2 Not descriptive variable name MIN_REQUEST_WITHDRAW_RANGE

2.4.3 Not descriptive parameter name _newMinRequestWithdrawRange

2.4.4 Unused variable DEFAULT_COMMISSION_RATE

2.4.5 Usage of deprecated OpenZeppelin's function _setupRole

2.4.6 Duplicate require statement in removeInvalidNodeOperator

2.4.7 Not descriptive function name setCommissionRate

2.4.8 Usage of UPPERCASE for name of non constant variable DEFAULT_COMMISSION_RATE

2.4.9 userHasRole modifier copies a OpenZeppelin's onlyRole functionality

2.4.10 StMATIC default admin is not DAO

2.4.11 Wrong @notice for addNodeOperator

2.4.12 No incentive to call removeInvalidNodeOperator

2.4.13 Only 2 functions in NodeOperatorRegistry.sol use whenNotPaused modifier

2.4.14 No zero check for _newMaxWithdrawPercentagePerRebalance parameter

2.4.15 Inaccurate comment for listDelegatedNodeOperators function

2.4.16 Typo in comment for _getValidatorsDelegationInfos

2.4.17 Not descriptive names for return variables in _getValidatorsDelegationInfos

2.4.18 Not descriptive variable name length

2.4.19 Excessive complexity of logical expressions in _getValidatorsDelegationInfos

2.4.20 Bad naming for bool variable

2.4.21 Reading validatorIds from storage twice

2.4.22 Magic number for precision

2.4.23 Not descriptive function name getValidatorsDelegationAmount

2.4.24 Misleading parameter name _totalBuffered in getValidatorsDelegationAmount

2.4.25 Not descriptive name for return variable validators in _getValidatorsDelegationInfos

2.4.26 Ambiguous term for validators

2.4.27 Not descriptive name for return variable totalActiveNodeOperator in getValidatorsDelegationAmount
2.4.28 Usage of comment instead of self-explanatory code .......................................................... 36
2.4.29 Complex calculations without intermediate variables in getValidatorsDelegationAmount .......................................................................................................................... 36
2.4.30 Misleading parameter name _totalBuffered in getValidatorsRebalanceAmount .......... 38
2.4.31 Not descriptive names for return variables in getValidatorsRebalanceAmount .......... 38
2.4.32 Division by zero if totalActiveNodeOperator is zero .......................................................... 39
2.4.33 Wrong tabulation .................................................................................................................. 40
2.4.34 getValidatorsRebalanceAmount can return zero ............................................................... 40
2.4.35 Inaccurate variable name activeValidators ........................................................................... 41
2.4.36 Inaccurate function name getValidatorsRequestWithdraw .................................................. 41
2.4.37 Singular form in variable name totalValidatorToWithdrawFrom ......................................... 42
2.4.38 Non usage of OpenZeppelin's Math utility contract in NodeOperatorRegistry.sol ........ 42
2.4.39 Default value of DISTANCE_THRESHOLD leads to unbalanced state ................................. 43
2.4.40 fxStateRootTunnel is not updated on each stMatic/matic rate update ............................... 44
2.4.41 Unclear variable name ......................................................................................................... 44
2.4.42 Variable name does not show that it's deprecated ............................................................... 45
2.4.43 RequestWithdraw name is confusing ................................................................................ 45
2.4.44 protocolFee is not initialized .............................................................................................. 46
2.4.45 Deprecated variables are set in initialize ............................................................................. 46
2.4.46 Typo in variable name totalValidatorToWithdrawFrom ..................................................... 47
2.4.47 Unclear variable name ......................................................................................................... 47
2.4.48 Redundant if ........................................................................................................................ 48
2.4.49 Math.min may increase readability ..................................................................................... 49
2.4.50 requestWithdraw may withdraw a little less than requested ............................................. 49
2.4.51 Duplicated storage read ..................................................................................................... 50
2.4.52 ValidatorData struct name is ambiguous ........................................................................... 50
2.4.53 Misleading variable names .................................................................................................. 51
2.4.54 Typo .................................................................................................................................... 51
2.4.55 Unnecessary nesting increases code complexity ................................................................. 52
2.4.56 buyVoucher may be called with 0 amount ....................................................................... 52
2.4.57 Variable names are too similar .......................................................................................... 53
2.4.58 Uninitialized local variables

2.4.59 Confusing variable name

2.4.60 rewardDistributionLowerBound is not initialized

2.4.61 Plural variable name that holds single value

2.4.62 Unused private function

2.4.63 setVersion does not emit an event

2.4.64 Trying to withdraw very small amount will burn requested tokens

2.4.65 withdrawalDelay used where it may be skipped

2.4.66 A validator may keep the system unbalanced

2.4.67 Unnecessary decreased readability

2.4.68 Erroneous comment

2.4.69 Magic numbers are used

2.4.70 Possible lock of the protocol if stMatic/matic rate is very big

2.4.71 Possible lock of protocol if withdrawExchangeRate is high

2.4.72 convertStMaticToMatic should be declared external

2.4.73 Active is used in several meanings

2.4.74 validatorRewardAddressTold is not reset on setRewardAddress

2.4.75 Copying storage validatorIds to memory has no point

2.4.76 Term ‘validator’ has different meanings throughout the codebase

2.4.77 Unnecessary nesting

2.4.78 totalValidatorToWithdrawFrom formula does not follow the docs

2.4.79 Mutating a variable instead of using several

2.4.80 Possibly undesired withdrawal proportions

2.4.81 Dangerous calculation

2.4.82 Reusing a variable in for-loop reduce readability

2.4.83 DISTANCE_THRESHOLD read several times from storage

2.4.84 Issues from report for PR#69 are not fixed here

2.4.85 Redundant check

2.4.86 Abstruse code

2.4.87 owner2Tokens[from]'s length does not decrease on a transfer

2.4.88 Forgotten import "hardhat/console.sol"
1 Introduction

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on this audit. Each of them check the provided source code independently of each other in accordance with the security assessment methodology described below:

1. Project architecture review:

Manually code study of the architecture of the code based on the source code only to find out the errors and bugs.

2. Check the code against the list of known vulnerabilities

Verification process of the code against the constantly updated list of already known vulnerabilities maintained by the company.

3. Architecture and structure check of the security model

Study project documentation and its comparison against the code including the study of the comments and other technical papers.

4. Result's cross-check by different auditors

Normally the research of the project is made by more than two auditors. After that, there is a step of the mutual cross-check process of audit results between different task performers.

5. Report consolidation

Consolidation of the audited report from multiple auditors.

6. Reaudit of new editions

After the client’s review and fixes, the founded issues are being double-checked. The results are provided in the new audit version.

7. Final audit report publication
The final audit version is prepared and provided to the client and also published on the official website of the company.

### 1.2.1 Severity Level Reference

Findings discovered during the audit are classified as follows: Every issue in this report was assigned a severity level from the following:

- **CRITICAL**: A bug leading to assets theft, fund access locking, or any other loss of funds due to transfer to unauthorized parties.
- **MAJOR**: A bug that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
- **WARNING**: A bug that can break the intended contract logic or expose it to DDoS attacks.
- **INFO**: Minor issue or recommendation reported to / acknowledged by the client's team.

### 1.2.2 Status Level Reference

Based on the feedback received from the client's team regarding the list of findings discovered by the contractor, the following statuses were assigned to the findings:

- **NEW**: Waiting for the project team's feedback.
- **FIXED**: Recommended fixes have been made to the project code and the identified issue no longer affects the project's security.
- **ACKNOWLEDGED**: The project team is aware of this finding. Recommended fixes for this finding are planned to be made. This finding does not affect the overall security of the project.
- **NO ISSUE**: Finding does not affect the overall security of the project and does not violate the logic of its work.
- **DISMISSED**: The issue or recommendation was dismissed by the client.

### 1.3 Project overview

Lido on Polygon is a DAO governed liquid staking protocol for the Polygon PoS chain. It allows users to stake their MATIC tokens on the Ethereum mainnet and immediately get the representation of their share in the form of stMATIC token without maintaining staking infrastructure. Users will get staking rewards and still control and utilize their stMATIC tokens in secondary markets on Ethereum mainnet and Polygon.

### 1.4 Audit Scope

The scope of the audit includes the following smart contracts at:

- `StMatic.sol`
- `NodeOperatorRegistry.sol`
- `PoLidoNFT.sol`
- `FxStateChildTunnel.sol`
- `FxStateRootTunnel.sol`
- `RateProvider.sol`
The audited commit identifier is 6b18e23ae258ff0aa84aecb82d8498f3c52f29e4

1.5 Steps to upgrade Lido on Polygon

The following plan is supposed by Lido on Polygon team to upgrade from current V1 version to V2 on the mainnet.

1.5.1 Contracts

- Deploy NodeOperatorRegistry V2
- Upgrade StMatic V1 to V2
- Upgrade LidoNFT V1 to V2

1.5.2 Configuration

StMatic

- Set stMatic nodeOperator address.
- Set stMatic ProtocolFee to 10%.
- Set stMatic version to 2.0.0.
- Set Role PAUSE && UNPAUSE roles.
- Upgrade Script

LidoNFT

- Set LidoNFT version to 2.0.0.
- Upgrade Script

NodeOperatorRegistry

- Add node operators.
- Deploy Script
- Upgrade Script
2 Report

2.1 CRITICAL
No critical issues found

2.2 MAJOR
No major issues found

2.3 WARNING

2.3.1 Incorrect `totalStaked` calculation

<table>
<thead>
<tr>
<th>Severity</th>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#L486

To calculate the `totalStake` value protocol uses operators with statuses ACTIVE and JAILED. But for calculation of `rebalanceTarget` `totalStaked` value is divided by number of active operators without considering jailed ones - NodeOperatorRegistry.sol#L562

It leads to incorrect `rebalanceTarget` and `distanceThreshold` values.

Let's consider an example with this initial conditions:

1. There are 4 nodes:
   - stake = 100, Status ACTIVE, delegate = false
   - stake = 40, Status ACTIVE, delegate = true
   - stake = 300, Status ACTIVE, delegate = true
   - stake = 250, Status ACTIVE, delegate = false
2. `totalStaked` = 100 + 40 + 300 + 250 = 690
3. `totalActiveNodeOperator` = 2
4. `stakePerOperator` = [40, 300]
5. `distanceThreshold` = (300 * 100) / 40 = 750

Calling the function `getValidatorsRebalanceAmount`:

1. Calculate `rebalanceTarget`: solidity `uint256 rebalanceTarget = totalStaked / totalActiveNodeOperator; // rebalanceTarget = 690 / 2 = 345`
2. Calculate `operatorRatioToRebalance` for each operator:

```solidity
operatorRatioToRebalance = stakePerOperator[idx] > rebalanceTarget ? stakePerOperator[idx] - rebalanceTarget : 0; // 40 > 345 == FALSE => operatorRatioToRebalance = 0 // 300 > 345 == FALSE => operatorRatioToRebalance = 0
```

`operatorRatioToRebalance` would always be a zero.

3. `totalRatio` would also be a zero too - (NodeOperatorRegistry.sol#L640)

4. `totalToWithdraw` would be a zero too so function would revert (NodeOperatorRegistry.sol#L646)

```solidity
totalToWithdraw = totalRatio > _totalBuffered ? totalRatio - _totalBuffered : 0; // totalToWithdraw = 0 require(totalToWithdraw > 0, "Zero total to withdraw");
```

**Recommendation**

To increase the `totalStake` value only for the operators with status `ACTIVE` and delegation.

**Update**

**Oxorio:**

We see that you ignore validators with disabled delegation there. So the main concern is that `totalStaked` is calculated for a bigger set of validators than `activeOperatorCount`.

`totalStaked` = Sum(ActiveWithDelegationOn + Jailed + ActiveWithDelegationOff).

`activeOperatorCount` = Count(ActiveWithDelegationOn).

And when you calculate `rebalanceTarget` you divide sum of funds from the bigger set by count of members of the smaller set. Because they are different sets this calculation is suspicious.

And validators may manipulate `activeOperatorCount` by enabling/disabling delegation.

Because they are able to manipulate denominator (`activeOperatorCount`) they can manipulate `rebalanceTarget`. But they can only make it bigger.

Because `rebalanceTarget` is manipulated to be big `stakePerOperator[idx] >= rebalanceTarget` will be false where it should not be. If enough validators or a validator with a big delegated stake participate in an attack it's possible to make it be equal to false all the time.

E.g. 3 validatos with stakes [1000, 600, 10].

First one disables delegation.

TotalStake = 1610.

`activeOperatorCount` = 2.

`rebalanceTarget` = 1610/2 = 805.

But this target is incorrect because it will try to rebalance to [1000, 805, 805] and not to 1610/3 = [536, 536, 536]
Even so the second one is already big enough the system will still try to delegate to it. As we said it's not a big deal just wanted you to know.

Almost the same logic apply to rebalanceDelegatedTokens->getValidatorsRebalanceAmount- >_getValidatorsDelegationInfos. But it's already described in the first report for V2. It's also not a big deal because you may indeed just remove a validator who behaves that way.

Shard Labs:

About the validators disable delegation, I believe that the best solution is to monitor the validators delegation state, then the DAO can remove the validator if it's required.

Oxorio:

We think that having a monitoring is a good solution. The issue is not that significant. We don't see how an attacker can use it to do any damage except unbalancing the stake.

2.3.2 Rounding error in $totalValidatorToWithdrawFrom$ calculation

<table>
<thead>
<tr>
<th>Severity</th>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#L771

```
totalValidatorToWithdrawFrom = ((withdrawAmountPercentage + MIN_REQUEST_WITHDRAW_RANGE) / (100 / length)) + 1;
```

This formula uses double division which can lead to rounding errors.

Also if $length > 100$ then $(100 / length) = 0$, so it leads to division by zero.

Recommendation

To change formula to

```
totalValidatorToWithdrawFrom = (withdrawAmountPercentage + MIN_REQUEST_WITHDRAW_RANGE) * length / 100 + 1;
```

Update

Fixed as recommended.
2.3.3 Reentrancy in `addNodeOperator` function

<table>
<thead>
<tr>
<th>Severity</th>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

NodeOperatorRegistry.sol#L133

If `validator.contractAddress` would have `ADD_NODE_OPERATOR_ROLE` role it can reenter to `addNodeOperator` and add several validators with the same id and different reward addresses. After this `validatorIds` array would have duplicated values which leads to incorrect calculations of different view functions.

**Recommendation**

To add `nonReentrant` modifier to `addNodeOperator` function

**Update**

*Fixed* as recommended.

2.3.4 DAO’s admin role not changing

<table>
<thead>
<tr>
<th>Severity</th>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

StMATIC.sol#L987

`revokeRole` must be called by `roleAdmin` but `dao` may not be DAO’s `roleAdmin`. Because DAO’s `roleAdmin` is not set it is `DEFAULT_ADMIN_ROLE`.

Right now `dao` has role `DEFAULT_ADMIN_ROLE`, but if it would change a new `dao` won't be able to call `setDaoAddress`.

**Recommendation**

To consider using `_revokeRole` or `renounceRole`

**Update**

*Fixed* by removing `revokeRole` and `_setupRole` from `setDaoAddress`.
Oxorio:

A little confusing, please consider updating the comment and maybe add that it changes dao address for rewards and does nothing to dao role. Consider renaming a variable `dao` to `daoRewardAddress`.

Shard Labs:

We prefer not changing the naming between the V1 and V2

### 2.3.5 `reservedFunds` may become greater than expected

<table>
<thead>
<tr>
<th>Severity</th>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

#### Description

`StMATIC.sol#L224`

`reservedFunds` can become greater than it should, even more than `totalBuffered`.

Prerequisites: equal stake across all validators.

The same is for 100% withdrawal with prerequisites above

1. requesting to withdraw > `totalDelegated`, let's say `totalDelegated + 1`
2. let's say `totalBuffered` is 5
3. let's say `totalValidatorToWithdrawFrom` is 10, rounding error will be 9
4. `currentAmount2WithdrawInMatic` returns 9 at worst case, even so it should be 1
5. `reservedFunds` += 9 (`StMATIC.sol#L266`)

As result: - **delegate** won't work until a **submit** because `reservedFunds > totalBuffered - _getTotalPooledMatic` will return incorrect value, less, that will lead to wrong exchange rate (more matic for stMatic that it should) So an attacker can withdraw more amount than it's available And only new submit will fix it, but it will also have incorrect exchange rate, less stMatic - it may also lead to DOS of all function that use `convert*` functions because of underflow - **rebalance** may also revert because of underflow

#### Recommendation

To update the code taking into account this case

#### Update

**Fixed** as recommended.
2.3.6 Possible erroneous conversion rate from Matic to StMatic

<table>
<thead>
<tr>
<th>Severity</th>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

_StMATIC.sol#L935_ If \(_\text{maticAmount} \times \text{totalStMaticSupply} < \text{totalStMaticSupply}\)_ then conversion rate would be erroneous because of rounding error.

Example: a lot of slashing happened, matic to stMatic = 2:1. No reserved funds. If someone requests \(\text{convertMaticToStMatic}(1)\) it will return 0, which is a wrong rate. \(\text{submit(1)}\) will mint 0.

Recommendation

To fix conversion Matic to StMatic in case then \(_\text{maticAmount} \times \text{totalStMaticSupply} < \text{totalStMaticSupply}\)

Update

_Fixed_ as recommended.

2.3.7 Possible erroneous conversion rate from StMatic to Matic

<table>
<thead>
<tr>
<th>Severity</th>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

_StMATIC.sol#L888_

If \(_\text{stMaticAmount} \times \text{totalPooledMatic} < \text{totalStMaticSupply}\)_ then conversion rate would be erroneous because of rounding error.

Example:

1. Start of contract, 100*10=100 matic is deposited, 100 stMatic given
2. delegate is not called
3. 900 is withdrawn, 900 is reserved, \(_\text{totalPooledMatic} = 100\)
4. Then one wants to withdraw 9 matic, so \(_\text{amountInMatic} = 9 \times 100 / 100 = 0\) (withdraw < 10)
As result requestWithdrawal creates a request with 0 amount, user lost his funds and burned stMatic.

**Recommendation**

To fix conversion StMatic to Matic in case then _stMaticAmount * _totalPooledMatic < totalStMaticSupply

**Update**

*Fixed* as recommended.

### 2.3.8 No check for creating withdraw request with a zero shares which leads to DOS

<table>
<thead>
<tr>
<th>Severity</th>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

*StMATIC.sol#L599*

It’s possible to create a withdraw request with zero shares. And then it will stuck forever consuming gas for calculatePendingBufferedTokens.

As result unstakeClaimTokens_new() would revert because it is requires that shares > 0 at *ValidatorShare.sol#L300*. So claimTokensFromValidatorToContract() and rebalanceDelegatedTokens() also would revert calling unstakeClaimTokens_new().

**Example:**

1. calls a withdrawTotalDelegated(), for example, stakedAmount = 1
2. _createWithdrawRequest(_validatorShare, stakedAmount);
3. mint new nft and create sellVoucher_new(_validatorShare, amount, type(uint256).max);
4. *StMATIC.sol#L636*
5. call matic contract =>
   
   IVendorShare(_validatorShare).sellVoucher_new(_claimAmount, _maximumSharesToBurn);
6. *StMATIC.sol#L774*
7. *ValidatorShare.sol#L238*
8. in function sellVoucher_new(), call _sellVoucher() (uint256 shares, uint256 _withdrawPoolShare) = _sellVoucher(claimAmount, maximumSharesToBurn);
9. create _sellVoucher() *ValidatorShare.sol#L274*
10. uint256 shares = claimAmount.mul(precision).div(rate);
11. if(claimAmount.mul(precision) < rate) => shares = 0
12. when we call function unstakeClaimTokens_new() =>
   IValidatorShare(_validatorShare).unstakeClaimTokens_new(uint256 unbondNonce)
13. in _unstakeClaimTokens() we call _unstakeClaimTokens() solidity require( unbond.withdrawEpoch.add(stakeManager.withdrawalDelay()) <= stakeManager.epoch() && shares > 0, "Incomplete withdrawal period" ); shares = 0 => function reverts

**Recommendation**

To add check for withdraw requests with zero shares

**Update**

*Fixed* as recommended.

### 2.3.9 Withdraw requests that are never used and consume gas

<table>
<thead>
<tr>
<th>Severity</th>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

*StMATIC.sol#L673*

If a user creates a requestWithdraw and sends it to the contract nothing can be done with that token. It would always consume gas in the calculatePendingBufferedTokens because user requests are saved in token2WithdrawRequest and token2WithdrawRequests mappings.

**Recommendation**

To fix code covering that case

**Update**

*Fixed* as recommended.

### 2.3.10 Revert of _requestWithdrawBalanced in case of inactive operators

<table>
<thead>
<tr>
<th>Severity</th>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>
Description

StMATIC.sol#L298

If operator in `activeNodeOperators[idx]` is inactive then function would revert because `activeNodeOperators[idx]` value would be empty. The only possible fix for this would be removal of the inactive operator by DAO.

It's not clear how to become inactive but your code suggest that it's possible.

```solidity
uint256 length = validatorIds.length;
activeValidators = new ValidatorData[](length);
```

`activeValidators`'s length is equal to `validatorIds`'s length. So for inactive validator ids `activeValidators` values would be empty.

Recommendation

To update the code for case of inactive operators

Update

Fixed as recommended.

2.3.11 Impossible to rebalance system if there are pending buffered tokens

<table>
<thead>
<tr>
<th>Severity</th>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

StMATIC.sol#L608

There is no need to add `calculatePendingBufferedTokens` to `amountToReDelegate` because it's impossible to delegate such tokens as they are pending.

So if there are pending buffered tokens in protocol then `amountToReDelegate` value would be greater than expected. As a result in `getValidatorsRebalanceAmount()` we get wrong calculation of `totalToWithdraw` at `NodeOperatorRegistry.sol#L642`: 
totalToWithdraw = totalRatio > _totalBuffered
? totalRatio - _totalBuffered
: 0;

totalRatio here does not include pending tokens, so totalToWithdraw would be 0 and it leads to the impossibility to rebalance until pending tokens are claimed.

So it’s impossible to rebalance more often than withdrawalDelay (~10 days) because each rebalance create one or several withdrawal requests.

**Recommendation**

To not include pending tokens to amountToReDelegate value

**Update**

**Shard Labs:**

We decided to include the calculatePendingBufferedTokens because when we delegate we also rebalance the system, in this case the system will be rebalanced using the calculatePendingBufferedTokens amount. Also, the withdrawalDelay (~3-4 days).

**Oxorio:**

The main point is that rebalanceDelegatedTokens will do nothing if there are pending tokens. If this is an expected behavior consider adding a check in the beginning to be clear, something like

```javascript
if (calculatePendingBufferedTokens() > 0) {
    return;
}
```

Because right now the function does that just with extra steps. And its behavior is not clear from the code.

**Shard Labs:**

This is normal behaviour if the calculatePendingBufferedTokens() has some value but it can not cover the rebalance we should be able to call rebalance function.
2.3.12 Possible burning Matic without minting shares during \texttt{delegate()}

<table>
<thead>
<tr>
<th>Severity</th>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

\texttt{StMATIC.sol#L411}

If \texttt{ValidatorShare.exchangeRate()} is low enough and \texttt{amountToDelegatePerOperator} is also low it's possible that matic tokens will be sent but no \texttt{ValidatorShare.shares} will be printed.

\texttt{ValidatorShare.sol#L379}

\begin{verbatim}
uint256 shares = _amount.mul(precision).div(rate);
\end{verbatim}

so you will effectively burn some tokens, because \texttt{_minSharesToMint} argument is zero.

If exchange rate is very high, e.g. 1:10^(18+3) calling \texttt{delegate} may burn quite a lot (~ 1000 matic).

It may happen if one validator was slashed a lot and then someone calls \texttt{delegate} here or if a single user is staked on this validator and the user withdraws ~100%, because of rounding error less shares will be burned and 1 matic will cost more shares.

**Recommendation**

To rewrite the logic for cover this case

**Update**

Fixed as recommended.

2.3.13 Attackers can make \texttt{calculatePendingBufferedTokens()} fail with out-of-gas

<table>
<thead>
<tr>
<th>Severity</th>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

\texttt{StMATIC.sol#L650}
The same major bug as was described in Oxorio audit for LidoV1 PR67 but it's also used in rebalance, so it's possible to lock this functions: - rebalanceDelegatedTokens - _getTotalPooledMatic - requestWithdraw - getTotalPooledMatic - claimTokensFromValidatorToContract - convertStMaticToMatic - convertMaticToStMatic - submit

But for claimTokensFromValidatorToContract nothing can be done after because claimTokensFromValidatorToContract requires token2WithdrawRequest but a user may send one with several requests token2WithdrawRequest*s*.

Recommendation

To update the code adding protection from such attacks

Update

Fixed as recommended.

2.3.14 Rounding error in amountToWithdraw calculation for operators with a small stake

<table>
<thead>
<tr>
<th>Severity</th>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

StMATIC.sol#L624

If totalRatio > operatorRatios[i] * totalToWithdraw when amountToWithdraw would be a zero.

```
operatorRatios[i] = operatorSharePercent * totalRatio
totalToWithdraw = maxWithdrawPercent * totalRatio
```

Let's transform initial inequality using this values:

```
totalRatio > operatorSharePercent * totalRatio * maxWithdrawPercent * totalRatio
1 > operatorSharePercent * totalRatio * maxWithdrawPercent
totalRato < 1 / (operatorSharePercent * maxWithdrawPercent)
totalRatio < 1 / operatorSharePercent / 5
```

```
So if \( \text{totalRatio} < \frac{5}{\text{operatorSharePercents}} \) then \( \text{amountToWithdraw} \) would be a zero.

Also because of rounding when \( \text{totalToWithdraw} \) is small it will require even less share percents to be rounded to 0 here (or more \( \text{totalRatio} \)).

**Recommendation**

To update the code covering that case

**Update**

*Fixed* as recommended.

### 2.4 INFO

#### 2.4.1 Not descriptive variable name

**DISTANCE_THRESHOLD**

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

*NodeOperatorRegistry.sol#L40*

```solidity
type uint256 public DISTANCE_THRESHOLD;
```

The variable's name does not show that it's value is a percent.

**Recommendation**

To rename \( \text{DISTANCE_THRESHOLD} \) to \( \text{DISTANCE_THRESHOLD_PERCENTS} \).

**Update**

*Fixed* as recommended.

#### 2.4.2 Not descriptive variable name

**MIN_REQUEST_WITHDRAW_RANGE**

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Severity** INFO

**Status** FIXED
The variable's name does not show that it's value is a percent.

**Recommendation**

To rename `MIN_REQUEST_WITHDRAW_RANGE` to `MIN_REQUEST_WITHDRAW_RANGE_PERCENTS`.

**Update**

Fixed as recommended.

### 2.4.3 Not descriptive parameter name

**newMinRequestWithdrawRange**

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

The parameter's name does not show that it's value is a percent. Also function's name does not show that it expects a percent value.

**Recommendation**

To rename function and its parameter: - `setMinRequestWithdrawRange` to `setMinRequestWithdrawRangePercents` - `newMinRequestWithdrawRange` to `newMinRequestWithdrawRangePercents`.

**Update**

Fixed as recommended.

### 2.4.4 Unused variable

**DEFAULT_COMMISSION_RATE**

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>
**Description**

*NodeOperatorRegistry.sol#L50*

```solidity
uint8 public DEFAULT_COMMISSION_RATE;
```

The variable is not used anywhere in the code except setter function `setCommissionRate`.

**Recommendation**

To remove `DEFAULT_COMMISSION_RATE` variable.

**Update**

Fixed as recommended.

**2.4.5 Usage of deprecated OpenZeppelin's function _setupRole**

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

*NodeOperatorRegistry.sol#L88*

```solidity
_setUserRole(DEFAULT_ADMIN_ROLE, msg.sender);
_setUserRole(PAUSE_ROLE, msg.sender);
_setUserRole(UNPAUSE_ROLE, _dao);
_setUserRole.DAO_ROLE, _dao);
_setUserRole(ADD_NODE_OPERATOR_ROLE, _dao);
_setUserRole(REMOVE_NODE_OPERATOR_ROLE, _dao);
```

OpenZeppelin's documentation says that `_setupRole` function is deprecated in favor of `_grantRole`.

**Recommendation**

To use `_grantRole` as OpenZeppelin's documentation recommends.

**Update**

Fixed [here](#) and [here](#) as recommended.
2.4.6 Duplicate require statement in removeInvalidNodeOperator

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#L190

```
require(rewardAddress != address(0), "Validator doesn't exist");
```

The same require check is already exists in the function 
_getOperatorStatusAndValidator at NodeOperatorRegistry.sol#L896

Recommendation

To remove duplicate require statement from removeInvalidNodeOperator.

Update

Fixed as recommended.

2.4.7 Not descriptive function name setCommissionRate

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#L235

The function name does not fully explain it's purpose because it's changing default commission rate not commission rate.

Recommendation

To rename setCommissionRate to setDefaultCloseOperationRate to be crystal clear

Update

The function is removed.
2.4.8 Usage of UPPERCASE for name of non constant variable DEFAULT_COMMISSION_RATE

Severity | INFO
Status    | FIXED

Description

NodeOperatorRegistry.sol#L245

```solidity
uint256 oldCommissionRate = DEFAULT_COMMISSION_RATE;
DEFAULT_COMMISSION_RATE = _newCommissionRate;
```

According to style guides uppercase names should be used only for constant variables but DEFAULT_COMMISSION_RATE value can be changed in setCommissionRate.

Recommendation

To use lowercase name for DEFAULT_COMMISSION_RATE

Update

The variable is removed.

2.4.9 userHasRole modifier copies a OpenZeppelin's onlyRole functionality

Severity | INFO
Status    | FIXED

Description

NodeOperatorRegistry.sol#L65

```solidity
modifier userHasRole(bytes32 _role) {
    require(hasRole(_role, msg.sender), "Unauthorized");
    _;
}
```

Recommendation

To remove userHasRole modifier and use onlyRole from OpenZeppelin's AccessControlUpgradeable instead.
Update

**Fixed** as recommended.

### 2.4.10 StMATIC default admin is not DAO

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>ACKNOWLEDGED</td>
</tr>
</tbody>
</table>

**Description**

*StMATIC.sol#L125*

```solidity
_setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
```

Now contract deployer gets default admin role so it may change dao or any other role.

**Recommendation**

To consider granting `DEFAULT_ADMIN_ROLE` to `_dao` not to `msg.sender`

**Update**

Shard Labs:

The initialize function can not called a second time (The contract will be upgraded only).

### 2.4.11 Wrong @notice for addNodeOperator

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

*NodeOperatorRegistry.sol#L98*

```solidity
/// @notice Add a new node operator to the system.  
/// ONLY DAO can execute this function.  
/// @param _validatorId the validator id on stakeManager.  
/// @param _rewardAddress the reward address.  
function addNodeOperator(uint256 _validatorId, address _rewardAddress)  
    external
```
Notice says that only DAO can execute this function but actually it's checks ADD_NODE_OPERATOR_ROLE not DAO_ROLE.

Recommendation

To update notice

Update

Fixed as recommended.

2.4.12 No incentive to call `removeInvalidNodeOperator`

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

`NodeOperatorRegistry.sol#L183`

There is no motivation to call this function, so protocol may need a centralized entity to control it.

Recommendation

To consider adding incentive to call `removeInvalidNodeOperator` or manage a centralized entity for this purpose

2.4.13 Only 2 functions in `NodeOperatorRegistry.sol` use `whenNotPaused` modifier

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

`NodeOperatorRegistry.sol#L183 NodeOperatorRegistry.sol#L269`

Only `removeInvalidNodeOperator` and `setRewardAddress` functions is pausable in a whole contract.
Recommendation

To make sure it's what you wanted and consider to remove pausability if not needed. Consider adding the info to the docs.

Update

Shard Labs:

Old code.

2.4.14 No zero check for \_newMaxWithdrawPercentagePerRebalance parameter

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#L323

If MAX_WITHDRAW_PERCENTAGE_PER_REBALANCE would set to zero it make rebalanceDelegatedTokens create several empty NFTs. Because totalToWithdraw will be 0.

Recommendation

To add zero check for \_newMaxWithdrawPercentagePerRebalance parameter

Update

Fixed here and here.

2.4.15 Inaccurate comment for listDelegatedNodeOperators function

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#L368
```solidity
/// @notice List all the ACTIVE operators on the stakeManager.
/// @return activeNodeOperators a list of ACTIVE node operator.
/// @return totalActiveNodeOperators total active node operators.
function listDelegatedNodeOperators()
```

This function returns not only ACTIVE operators as @notice says but also delegated ones.

Also "delegated" term in function's name is confusing because it has different meaning with "enabled delegation". For example operator can be delegated but with disabled delegation and it would not returned.

**Recommendation**

To update function's @notice. Also to consider using another term instead "delegated" e.g. activeWithEnabledDelegation.

**2.4.16 Typo in comment for _getValidatorsDelegationInfos**

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

NodeOperatorRegistry.sol#L441

```solidity
/// @return activeOperatorCount count onlt active validators.
```

"onlt" should be a "only".

**Recommendation**

To fix typo

**Update**

Fixed as recommended.
2.4.17 Not descriptive names for return variables in \_getValidatorsDelegationInfos

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#L449

```solidity
function \_getValidatorsDelegationInfos() private view returns (ValidatorData[] memory validators, uint256 activeOperatorCount, uint256[] memory stakePerOperator, uint256 totalStaked, uint256 distanceThreshold)
```

Return variable names `validators`, `activeOperatorCount` and `stakePerOperator` are not descriptive which reduces code readability.

Recommendation

To rename return variables to be more descriptive: - validators to `activeWithEnabledDelegationValidators` - `activeOperatorCount` to `activeWithEnabledDelegationOperatorCount` - `stakePerOperator` to `stakePerActiveWithEnabledDelegationOperator`

2.4.18 Not descriptive variable name `length`

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#L456  
NodeOperatorRegistry.sol#L767

`length` name is too generic.
Recommendation

To rename `length` to `validatorIdsLength`

### 2.4.19 Excessive complexity of logical expressions in `_getValidatorsDelegationInfos`

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO ISSUE</td>
</tr>
</tbody>
</table>

**Description**

NodeOperatorRegistry.sol#L473 NodeOperatorRegistry.sol#L478

```solidity
require(
  !(status == NodeOperatorRegistryStatus.EJECTED),
  "Could not calculate the stake data, an operator was EJECTED"
);
require(
  !(status == NodeOperatorRegistryStatus.UNSTAKED),
  "Could not calculate the stake data, an operator was UNSTAKED"
);
```

Such logical expressions as `!(x == y)` are uncommon and they are harder to read comparing to more common `x != y`. So it's reduces code readability.

**Recommendation**

To use simplify logical expressions

### 2.4.20 Bad naming for bool variable

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

NodeOperatorRegistry.sol#L496

```solidity
bool delegation = IValidatorShare(validator.contractAddress).delegation();
```
Name `delegation` doesn't capture its essence and doesn't show that it's a boolean variable so it reduces code readability.

**Recommendation**

To rename `delegation` to `isDelegationEnabled`.

**Update**

Fixed as recommended.

### 2.4.21 Reading `validatorIds` from storage twice

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>ACKNOWLEDGED</td>
</tr>
</tbody>
</table>

**Description**

*NodeOperatorRegistry.sol#L504*

```solidity
class NodeOperatorRegistry {
  // ... some code ...

  validators[activeOperatorCount] = ValidatorData(
      validator.contractAddress,
      validatorIdToRewardAddress[validatorIds[i]]
  );
```

No need to read `validatorIds[i]` from storage again as it's already stored in `validatorId` variable (*NodeOperatorRegistry.sol#L468*).

**Recommendation**

To use `validatorId` instead of reading from storage.

### 2.4.22 Magic number for precision

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>ACKNOWLEDGED</td>
</tr>
</tbody>
</table>

**Description**

*NodeOperatorRegistry.sol#515*

```solidity
distanceThreshold = ((maxAmount * 100) / minAmount);
```
100 is a magic number here. And it's repeated in other places where a precise value is needed.

Recommendation

To use a constant with a descriptive name here and other places for having precise values

2.4.23 Not descriptive function name
getValidatorsDelegationAmount

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#527

Function's name is not descriptive and doesn't show that function return information about active validators with enabled delegation.

Recommendation

To rename `getValidatorsDelegationAmount` to have more meaningful name, e.g.
`getInfoForDelegationOfActiveValidatorsWithEnabledDelegation`

2.4.24 Misleading parameter name `_totalBuffered` in
getValidatorsDelegationAmount

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#527

 `_totalBuffered` name is misleading because it is actually a `totalBuffered - reservedFunds`

Recommendation

To rename `_totalBuffered`, e.g. to `amountToDelegate` in
`getValidatorsDelegationAmount`

Update

Fixed as recommended.
2.4.25 Not descriptive name for return variable validators in _getValidatorsDelegationInfos

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#532

This variable named validators but it stores only active or jailed validators not all of them which is misleading.

Recommendation

To rename validators to activeOrJailedValidators

2.4.26 Ambiguous term for validators

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>ACKNOWLEDGED</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#532

In StMATIC.sol#L610 you named this variable nodeOperators but here it named validators which is misleading.

Recommendation

To make a dictionary of used terms and use the same terms in all contracts.

2.4.27 Not descriptive name for return variable totalActiveNodeOperator in getValidatorsDelegationAmount

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#533
This variable named `totalActiveNodeOperator` but it stores number of active or jailed validators not only active which is misleading and may result to a mistake.

Also `totalActiveNodeOperator` is a singular.

**Recommendation**

To rename `totalActiveNodeOperator` to `totalActiveOrJailedNodeOperators`.

### 2.4.28 Usage of comment instead of self-explanatory code

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

`NodeOperatorRegistry.sol#551`

```solidity
// If the system is balanced
if (distanceThreshold <= DISTANCE_THRESHOLD) {
```

Explicit is better than implicit. Introducing a new bool variable allows to remove comment and make code self-explanatory.

**Recommendation**

To remove comment and to add a bool variable and use it in `if` statement:

```solidity
bool isTheSystemBalanced = distanceThreshold <= DISTANCE_THRESHOLD
if (isTheSystemBalanced) {
```

**Update**

*Fixed* as recommended.

### 2.4.29 Complex calculations without intermediate variables in `getValidatorsDelegationAmount`

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Report
Description

NodeOperatorRegistry.sol#567

```solidity
for (uint256 idx = 0; idx < totalActiveNodeOperator; idx++) {
    operatorRatioToDelegate = stakePerOperator[idx] >=
    rebalanceTarget
    ? 0
    : rebalanceTarget - stakePerOperator[idx];

    if (operatorRatioToDelegate != 0 && stakePerOperator[idx] != 0) {
        operatorRatioToDelegate = (rebalanceTarget * 100) /
        stakePerOperator[idx] >=
        DISTANCE_THRESHOLD
        ? operatorRatioToDelegate
        : 0;
    }

    operatorRatios[idx] = operatorRatioToDelegate;
    totalRatio += operatorRatioToDelegate;
}
```

Introducing intermediate variables in this loop allows to greatly improve code readability and
decrease it’s complexity. Almost the same is valid for for-loop in 'getValidatorsRebalanceAmount'.

Recommendation

To consider introducing intermediate variables for better code readability. For example:

```solidity
for (uint256 idx = 0; idx < totalActiveNodeOperator; idx++) {
    bool hasMoreThanAvg = stakePerOperator[idx] >= rebalanceTarget;
    bool hasStake = stakePerOperator[idx] != 0;

    operatorRatioToDelegate = hasMoreThanAvg ? 0 : rebalanceTarget -
    stakePerOperator[idx];

    if (!hasMoreThanAvg && hasStake) {
        uint256 distance = (rebalanceTarget * 100) /
        stakePerOperator[idx];
        bool shouldDelegate = distance >= DISTANCE_THRESHOLD;
        operatorRatioToDelegate = shouldDelegate ?
        operatorRatioToDelegate : 0;
    }

    operatorRatios[idx] = operatorRatioToDelegate;
}
```
totalRatio += operatorRatioToDelegate;
}

2.4.30 Misleading parameter name `_totalBuffered` in `getValidatorsRebalanceAmount`

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#595

`_totalBuffered` name is misleading because it is actually a `totalBuffered - reservedFunds`.

Recommendation

To rename `_totalBuffered`, e.g. to `amountToReDelegate`.

Update

*Fixed* as recommended.

2.4.31 Not descriptive names for return variables in `getValidatorsRebalanceAmount`

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#600

```solidity
function getValidatorsRebalanceAmount(uint256 _totalBuffered) external view override returns (ValidatorData[] memory validators,
    uint256 totalActiveNodeOperator,
    uint256[] memory operatorRatios,
    uint256 totalRatio,
```
Return variables names validators and totalActiveNodeOperator are not descriptive which reduces code readability.

Also totalActiveNodeOperator is a singular.

**Recommendation**

To rename return variables to be more descriptive:

- validators to activeWithEnabledDelegationValidatorAddresses
- totalActiveNodeOperator to totalActiveWithEnabledDelegationNodeOperators

### 2.4.32 Division by zero if totalActiveNodeOperator is zero

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

NodeOperatorRegistry.sol#L607

```solidity
uint256 rebalanceTarget = totalStaked / totalActiveNodeOperator;
```

It's possible that validatorIds.length would be greater than totalActiveNodeOperator then this require is pass. In this case if totalActiveNodeOperator is zero then would be division by zero at NodeOperatorRegistry.sol#L625

**Recommendation**

To add zero check for totalActiveNodeOperator

**Update**

Fixed as recommended.
2.4.33 Wrong tabulation

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

NodeOperatorRegistry.sol#L612  NodeOperatorRegistry.sol#L543  https://github.com/Shard-Labs/PoLido-V2/blob/6b18e23ae258ff0aa84aecb82d8498f3c52f29e4/contracts/
NodeOperatorRegistry.sol#L553
NodeOperatorRegistry.sol#L756

The tabulation is wrong at this lines.

**Recommendation**

To fix wrong tabulation

**Update**

Fixed as recommended.

2.4.34 getValidatorsRebalanceAmount can return zero

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

NodeOperatorRegistry.sol#647

```solidity
require(totalToWithdraw > 0, "Zero total to withdraw");
totalToWithdraw = (totalToWithdraw * MAX_WITHDRAW_PERCENTAGE_PER_REBALANCE) / 100;
```

If totalToWithdraw * MAX_WITHDRAW_PERCENTAGE_PER_REBALANCE < 100 then totalToWithdraw would be a zero. As there is no zero checks for totalToWithdraw then in this case StMatic.rebalanceDelegatedTokens would create several empty NFTs. There will be no intention to claim them - users will pay for gas and get nothing. It will make calculatePendingBufferedTokens (and all the functions calling it) cost more.

Also totalToWithdraw can be zero if MAX_WITHDRAW_PERCENTAGE_PER_REBALANCE is set to 0.
Recommendation

To make zero check for `totalToWithdraw` before returning a value from function.

Update

**Fixed** as recommended.

### 2.4.35 Inaccurate variable name `activeValidators`

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

`NodeOperatorRegistry.sol#662`

Active validator is a validator with a status ACTIVE but actually this variable stores all validators exclude ones with status INACTIVE which is misleading.

Recommendation

To rename `activeValidators` to `nonInactiveValidators`

Update

**Fixed** as recommended.

### 2.4.36 Inaccurate function name `getValidatorsRequestWithdraw`

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

`NodeOperatorRegistry.sol#715`

The current function's name is inaccurate as it returns validators for request withdrawal.

Recommendation

To consider renaming `getValidatorsRequestWithdraw` to `getValidatorsForRequestWithdraw`
2.4.37 Singular form in variable name

totalValidatorToWithdrawFrom

Severity | INFO
Status | NO_ISSUE

Description

NodeOperatorRegistry.sol#L771

Word "validator" in variable name should be in a plural form.

Recommendation

To rename totalValidatorToWithdrawFrom to totalValidatorsToWithdrawFrom

2.4.38 Non usage of OpenZeppelin's Math utility contract in NodeOperatorRegistry.sol

Severity | INFO
Status | NO_ISSUE

Description

NodeOperatorRegistry.sol#L776

```
totalValidatorToWithdrawFrom = totalValidatorToWithdrawFrom > length
length
: totalValidatorToWithdrawFrom;
// totalValidatorToWithdrawFrom = Math.min(totalValidatorToWithdrawFrom, length)
```

NodeOperatorRegistry.sol#L804

```
rebalanceTarget = rebalanceTarget > minAmount
minAmount
: rebalanceTarget;
// rebalanceTarget = Math.min(rebalanceTarget, minAmount)
```

NodeOperatorRegistry.sol#L797
withdrawAmountPercentage = withdrawAmountPercentage == 0 ? 1 : withdrawAmountPercentage;
// withdrawAmountPercentage = Math.max(1, withdrawAmountPercentage);

Code using Math.min and Math.max from OpenZeppelin's Math contract is much easier to read and maintain.

Recommendation

To consider using OpenZeppelin's Math.min and Math.max utility functions to improve code readability.

2.4.39 Default value of DISTANCE_THRESHOLD leads to unbalanced state

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#L971

distanceThreshold = ((maxAmount * 100) / min);
isBalanced = distanceThreshold <= DISTANCE_THRESHOLD;

By default DISTANCE_THRESHOLD is 100. So it means that isBalanced would always be false by default.

Recommendation

To consider changing default value of DISTANCE_THRESHOLD.

Update

Fixed as recommended.
2.4.40 fxStateRootTunnel is not updated on each stMatic/matic rate update

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>ACKNOWLEDGED</td>
</tr>
</tbody>
</table>

**Description**

줄임소스(StMATIC.sol#L41)

```solidity
IFxStateRootTunnel public override fxStateRootTunnel;
```

fxStateRootTunnel may easily become out of sync.

fxStateRootTunnel is not updated on `withdrawTotalDelegated` call even so the call changes matic/stMatic rate because of matic rewards that are sent to the StMatic contract after the call. Also slashing or sending Matic directly to StMatic (using `transfer`) will update the rate on child chain, but not on root chain.

**Recommendation**

Consider making it clear in the docs and the comments that data from fxChild should not be trusted.

2.4.41 Unclear variable name

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

**Description**

줄임소스(StMATIC.sol#L53)

```solidity
address public override token;
```

It may be unclear which token it is.

**Recommendation**

Consider renaming to `maticToken` to be crystal clear which token it is.
2.4.42 Variable name does not show that it's deprecated

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

StMATIC.sol#L56

```solidity
class StMATIC {
    uint256 public override lastWithdrawnValidatorId;
}
```

lastWithdrawnValidatorId is not used but does not display it in its name. It may lead to erroneous usage of a variable in the future, as have happened with submitThreshold and submitHandler (See 'Deprecated variables are set in initialize').

Recommendation

Rename lastWithdrawnValidatorId to lastWithdrawnValidatorIdDeprecated.

Update

Shard Labs:

We prefer not changing the naming between the V1 and V2

2.4.43 RequestWithdraw name is confusing

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

StMATIC.sol#L76-77

```solidity
/// @notice token to WithdrawRequest mapping one-to-one.
mapping(uint256 => RequestWithdraw) public override
token2WithdrawRequest;
```

Usually variable names have a noun at the end. As you used in your comment. Or named it in token2WithdrawRequest.
Recommendation

Rename `RequestWithdraw` to `WithdrawRequest`.

Update

Shard Labs:

We prefer not changing the naming between the V1 and V2.

2.4.44 `protocolFee` is not initialized

| Severity | INFO |
| Status   | NO_ISSUE |

Description

`StMATIC.sol#L88`

```solidity
uint8 public override protocolFee;
```

It will lock `distributeRewards` until `protocolFee` is set. `StMATIC.sol#L542`

```solidity
uint256 totalRewards = ((IERC20Upgradeable(token).balanceOf(address(this)) - totalBuffered)) / protocolFee;
```

Recommendation

Consider adding `protocolFee` to `initialize`.

Update

Shard Labs:

We will upgrade the contract and `initialize` function will not run.

2.4.45 Deprecated variables are set in `initialize`

| Severity | INFO |
| Status   | FIXED |
Description

StMATIC.sol#L139

```
submitThreshold = _submitThreshold;
submitHandler = true;
```

Recommendation

Remove `submitThreshold` and `submitHandler` from `initialize` and rename to `submitThresholdDeprecated` and `submitHandlerDeprecated`.

Update

Fixed as recommended.

2.4.46 Typo in variable name

totalValidatorToWithdrawFrom

Severity | INFO
---|---
Status | FIXED

Description

StMATIC.sol#L198

```
uint256 totalValidatorToWithdrawFrom
```

It should be in plural form.

Recommendation

Rename to `totalValidatorsToWithdrawFrom`.

Update

Fixed as recommended.

2.4.47 Unclear variable name
Description

StMATIC.sol#L199

```solidity
uint256[] memory operatorAmountCanBeRequested
```

It hard to understand from `operatorAmountCanBeRequested` name what it's holding.

Recommendation

Consider renaming to `allowedAmountToRequestFromOperators`.

Update

Fixed as recommended.

2.4.48 Redundant if

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

StMATIC.sol#L207

```solidity
if (totalDelegated != 0) {
    require((totalDelegated + totalBuffered) >= totalAmount2WithdrawInMatic, "Too much to withdraw");
}
else {
    require(totalBuffered >= totalAmount2WithdrawInMatic, "Too much to withdraw");
}
```

Recommendation

Consider using

```solidity
require((totalDelegated + totalBuffered) >= totalAmount2WithdrawInMatic, "Too much to withdraw");
```

to simplify the code.
Update

**Fixed** as recommended.

### 2.4.49 Math.min may increase readability

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

**Description**

StMATIC.sol#L291

```solidity
uint256 totalAmount = totalDelegated > totalAmount2WithdrawInMatic ?
totalAmount2WithdrawInMatic : totalDelegated;
```

**Recommendation**

Math.min(totalDelegated, totalAmount2WithdrawInMatic)

---

### 2.4.50 requestWithdraw may withdraw a little less than requested

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>ACKNOWLEDGED</td>
</tr>
</tbody>
</table>

**Description**

_requestWithdrawBalanced may return a little bit less because of rounding error. Which may not be expected by a 3rd party. A 3rd party may expect that amount passed to requestWithdraw is returned exactly (which happens in most cases).

StMATIC.sol#L294

```solidity
uint256 amount2WithdrawFromValidator = totalAmount /
totalValidatorToWithdrawFrom;
```

**Recommendation**

Consider adding this info to docs for 3rd parties or rewriting so everything is withdrawn.
2.4.51 Duplicated storage read

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

StMATIC.sol#L370

```solidity
require(totalBuffered > delegationLowerBound + reservedFunds, "Amount to delegate lower than minimum");
uint256 amountToDelegate = totalBuffered - reservedFunds;
```

Reads totalBuffered and reservedFunds twice. You may move amountToDelegate declaration above require because totalBuffered is always >= reservedFunds. It will also increase readability because of reduced number of computations inside require statement.

Recommendation

```solidity
uint256 amountToDelegate = totalBuffered - reservedFunds;
require(amountToDelegate > delegationLowerBound, "Amount to delegate lower than minimum");
```

Update

Fixed as recommended.

2.4.52 ValidatorData struct name is ambiguous

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

INodeOperatorRegistry.sol#L42-L45

```solidity
struct ValidatorData {
    address validatorShare;
    address rewardAddress;
}
```

Data is an ambiguous word. The struct holds addresses.
Recommendation

Consider renaming `ValidatorData` to `ValidatorAddresses` or something more meaningful.

2.4.53 Misleading variable names

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

StMATIC.sol#L377

```solidity
INodeOperatorRegistry.ValidatorData[] memory activeNodeOperators
```

StMATIC.sol#L379

```solidity
uint256 totalActiveNodeOperator
```

A reader may expect that `activeNodeOperators` includes only ones with `ACTIVE` status. But in fact it also includes jailed ones.

Recommendation

Consider renaming to `activeOrJailedNodeOperators` or `delegatableNodeOperators`. And to `totalActiveOrJailedNodeOperators` or `totalDelegatableNodeOperators`.

Update

Fixed as recommended.

2.4.54 Typo

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

StMATIC.sol#L398
If the total Ratio is equal to ZERO that means the system is balanced so we recommend replacing with "system is".

Update

Fixed as recommended.

2.4.55 Unnecessary nesting increases code complexity

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

StMATIC.sol#L404

```solidity
} else {
    if (operatorRatios[i] == 0) continue;
    amountToDelegatePerOperator = (operatorRatios[i] * amountToDelegate) / totalRatio;
}
```

The less nestings the better.

Recommendation

Use else if(operatorRatios[i] == 0) and else.

2.4.56 buyVoucher may be called with 0 amount

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

StMATIC.sol#L406
amountToDelegatePerOperator = (operatorRatios[i] * amountToDelegate) / totalRatio;

The statement above may be 0 if operatorRatios[i] * amountToDelegate < totalRatio. Because of it, buyVoucher may be called with amountToDelegatePerOperator set to 0. Which will effectively just burn a lot of gas comparing to a check.

**Recommendation**

Consider adding `if(amountToDelegatePerOperator > 0)` before calling `buyVoucher`.

**Update**

*Fixed* as recommended.

### 2.4.57 Variable names are too similar

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

**Description**

`token2WithdrawRequest` and `token2WithdrawRequests` is hard to distinguish when you read the code. It may be confusing for a reader.

**StMATIC.sol#L432**

```solidity
if (token2WithdrawRequest[_tokenId].requestEpoch != 0) {
  _claimTokensV1(_tokenId);
} else if (token2WithdrawRequests[_tokenId].length != 0) {
  _claimTokensV2(_tokenId);
}
```

**Recommendation**

Consider renaming `token2WithdrawRequest` to `token2WithdrawRequestV1`

**Update**

*Shard Labs:*

We prefer not changing the naming between the V1 and V2
2.4.58 Uninitialized local variables

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

**Description**

StMATIC.sol#L453

```solidity
uint256 amountToClaim;
```

**Recommendation**

Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

2.4.59 Confusing variable name

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

**Description**

StMATIC.sol#L526

```solidity
uint256 totalActiveOperatorInfos
```

The variable name is totalActiveOperatorInfos but in fact it includes only active with enabled delegation.

**Recommendation**

Consider renaming `totalActiveOperatorInfos` to `totalActiveWithEnabledDelegationOperatorInfos` or `totalOperatorInfosToDistribute`.

2.4.60 rewardDistributionLowerBound is not initialized

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>
require(totalRewards > rewardDistributionLowerBound, "Amount to
distribute lower than minimum");

It will disable require above and allow to distribute any amount. Which is not the desired behavior.

### Recommendation
Consider initializing `rewardDistributionLowerBound` to a sane default value.

#### [NO_ISSUE] If `insurance` is set to `address(0)` `distributeRewards` will revert

#### Description
Default OZ ERC20 implementation has `require` that checks that `to` is not `address(0)`. If `insurance` is set to `address(0)` distribute rewards will revert on this line.

[StMATIC.sol#L561](https://github.com/Shard-Labs/PoLido-V2/blob/6b18e23ae258ff0aa84aebcb82d8498f3c52f29e4/contracts/StMATIC.sol#L561)

```
IERC20Upgradeable(token).safeTransfer(insurance, insuranceRewards);
```

Because of that `address(0)` has a side effect which may not be expected.

### Recommendation
Consider adding zero-address checks for `insurance` in `setInsuranceAddress` and `initialize`.

#### [ACKNOWLEDGED] `withdrawTotalDelegated` not always emit an event on success

#### Description
In this case event is not emitted. It just `returns`. It may be unexpected.

[StMATIC.sol#L595](https://github.com/Shard-Labs/PoLido-V2/blob/6b18e23ae258ff0aa84aebcb82d8498f3c52f29e4/contracts/StMATIC.sol#L595)
if (stakedAmount == 0) {
    return;
}

Recommendation
Consider emitting an event inside the if statement or writing about this behavior in the docs.

2.4.61 Plural variable name that holds single value

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description
StMATIC.sol#L679

RequestWithdraw storage lidoRequests =
token2WithdrawRequest[_tokenId];

`lidoRequests` holds a single request but the name suggests that it has multiple.

Recommendation
Rename `lidoRequests` to `lidoRequest`.

Update
Fixed as recommended.

2.4.62 Unused private function

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description
StMATIC.sol#L751

`function restake(address _validatorShare) private {`  
   `IVValidatorShare(_validatorShare).restake();`  
`}`
Restake is never used.

Recommendation
Remove restake function.

Update
Fixed as recommended.

2.4.63 setVersion does not emit an event

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description
StMATIC.sol#L1075

```{  
    version = _newVersion; 
}  
```

Subscribing to a version change may be useful for the protocol users.

Recommendation
Consider emitting an event on version change.

Update
Fixed as recommended.

2.4.64 Trying to withdraw very small amount will burn requested tokens

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description
When the system is balanced and a user requests a very small amount, less than `totalValidatorToWithdrawFrom` to withdraw there will be a lot of empty requests created because `amount2WithdrawFromValidator` is 0.
It may be more user-friendly to add a `require` that checks that amount is not 0. In case a user requested a wrong amount by mistake. Also it will burn the user's tokens and send message to child with wrong `totalPooledMatic` because `totalAmount2WithdrawInMatic` is not actually withdrawn. \texttt{StMATIC.sol#L272-L277}

```
fxStateRootTunnel.sendMessageToChild(
    abi.encode(
        totalSupply(),
        totalPooledMatic - totalAmount2WithdrawInMatic
    )
);
```

It will also emit an event that may break 3rd party accounting because `_amount` is not withdrawn.

**Recommendation**

Consider adding a `require` that checks `amount2WithdrawFromValidator > 0`.

**Update**

Fixed \texttt{here} and \texttt{here}.

\textbf{Oxorio}:

a. Wrong value passed to `_calculateValidatorShares`, should be `amount2WithdrawFromValidator` and `amount2WithdrawFromValidator` in both. b. in `_requestWithdrawBalanced` may start reverting on small values because share rate is different on each validator and one may have very small rate, that will return 0 shares for X amount. c. in `_requestWithdrawUnbalanced` may be impossible to withdraw more than half in one tx Because a small validator may have ~0 amount (but not 0) and it can be first in `smallNodeOperatorIds`

\textbf{Shard Labs}:

a. Fixed in \texttt{aa31000c58624b0631afdba6b858d70ba7dab715} b. NO-ISSUE this is an edge case, this can happen only if the validator was slashed a lot in this case DAO should unstake the delegation. c. NO-ISSUE this is an edge case, Users are aware about small number precision. Also the system will balance it self automatically. this can happen only if the validator is stopped and we are the last withdrawer.
2.4.65 **withdrawalDelay** used where it may be skipped

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

**Description**

*StMATIC.sol#L262*

```solidity
if (_amount > totalDelegated) {
    token2WithdrawRequests[tokenId].push(
        RequestWithdraw(
            currentAmount2WithdrawInMatic, 0,
            stakeManager.epoch() + stakeManager.withdrawalDelay(),
            address(0)
        )
    );
    reservedFunds += currentAmount2WithdrawInMatic;
    currentAmount2WithdrawInMatic = 0;
}
```

Here a user don't really have to wait for a withdrawal. We reserve funds that are already on the StMatic contract. We can give them right away.

**Recommendation**

Consider removing delay for requests that use matic from the contract.

2.4.66 **A validator may keep the system unbalanced**

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>ACKNOWLEDGED</td>
</tr>
</tbody>
</table>

**Description**

*StMATIC.sol#L368* A validator may disable delegation before a call to delegate and rebalanceDelegatedTokens to keep the system unbalanced. So one validator with disabled delegation will make a system spend gas on trying to make it balanced. An incentive for a validator to do that may be a "bribe" from a competitor. It will make usage of Lido more expensive. And you have to rely on DAO to remove that validator, which may take time.
Recommendation

Estimate the risks of this behavior and consider to implement a mechanism to discourage it

2.4.67 Unnecessary decreased readability

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

Introducing a variable may increase readability a lot.

StMATIC.sol#L474

```solidity
token).balanceOf(address(this)) - balanceBeforeClaim;
```

Recommendation

Add a variable

```solidity
uint256 balanceNow = IERC20Upgradeable(token).balanceOf(address(this));
amountToClaim += balanceNow - balanceBeforeClaim;
```

2.4.68 Erroneous comment

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

StMATIC.sol#L968

```solidity
/// @param _newProtocolFee - Insurance fee in %
```

Protocol fee is not 'Insurance fee in %'; it’s denominator that will be used in totalRewards calculation. StMATIC.sol#L542-L544
The calculation is also a bit confusing.

**Recommendation**

Update the comment. Consider using `protocolFee / 100` instead.

**Update**

*Fixed* as recommended.

---

**2.4.69 Magic numbers are used**

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

Magic numbers at *StMATIC.sol#L1113* decrease code readability. A reader won’t understand what they mean without context. Moreover, it complicates code maintenance.

```
uint256 exchangeRatePrecision = validatorId < 8 ? 100 : 10**29;
```

**Recommendation**

We recommend using constants with descriptive names or adding a comment explaining what is happening.

**Update**

*Fixed* as recommended.
2.4.70 Possible lock of the protocol if stMatic/matic rate is very big

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>ACKNOWLEDGED</td>
</tr>
</tbody>
</table>

**Description**

[StMATIC.sol#L888](#)

```
uint256 amountInMatic = (_stMaticAmount * _totalPooledMatic) / totalStMaticSupply;
```

It can only happen if stMatic/matic exchange rate is very big.

For example _totalPooledMatic is a billion full matics (10^27). type(uint256).max ~ 10^77. _stMaticAmount should be > 10^77/10^27 > 10^50.

**Recommendation**

Just make sure the exchange rate does not grow that much.

2.4.71 Possible lock of protocol if withdrawExchangeRate is high

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>ACKNOWLEDGED</td>
</tr>
</tbody>
</table>

**Description**

[StMATIC.sol#L1118](#)

```
return (withdrawExchangeRate * unbond.shares) / exchangeRatePrecision;
```

Possible overflow in rare cases when withdrawExchangeRate is very high. It can happen if a validorShare was slashed heavily and it will lock almost all the functions because _getMaticFromTokenId is used in the exchange rate calculation. It's possible if validator can somehow behave that way that it is slashed on desired big amount. So they are not slashed on 100%, but have a little matic left. Can be fixed by DAO removal of this validator.
Recommendation

Research if it's possible and act accordingly

2.4.72 \texttt{convertStMaticToMatic} should be declared external

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

\texttt{StMATIC.sol#L859-L860}

\texttt{function convertStMaticToMatic(uint256 \_amountInStMatic) public}

Recommendation

Public functions that are never called by the contract should be declared external to save gas.

Update

\textit{Fixed} as recommended.

2.4.73 \texttt{Active} is used in several meanings

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

\texttt{StMATIC.sol#L191}

\texttt{memory activeNodeOperators,}

Here it means "not having status 'NodeOperatorRegistryStatus.INACTIVE'". A reader may expect that it means "having status 'NodeOperatorRegistryStatus.ACTIVE'".

The same term is used in different meaning in another function, \texttt{listDelegatedNodeOperators}. There in means "having status 'NodeOperatorRegistryStatus.ACTIVE' and enabled delegation".

Report 63
Recommendation

Consider creating a terminology that you will use throughout the codebase. So `activeValidators` and other terms mean only one thing.

2.4.74 `validatorRewardAddressToId` is not reset on `setRewardAddress`

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

`NodeOperatorRegistry.sol#L269-L280`

```solidity
function setRewardAddress(address _newRewardAddress) whenNotPaused external override {
    uint256 validatorId = validatorRewardAddressToId[msg.sender];
    address oldRewardAddress = validatorIdToRewardAddress[validatorId];
    require(oldRewardAddress == msg.sender, "Unauthorized");
    require(_newRewardAddress != address(0), "Invalid reward address");

    validatorIdToRewardAddress[validatorId] = _newRewardAddress;

    emit SetRewardAddress(validatorId, oldRewardAddress, _newRewardAddress);
}
```

It leads to: 1. Inconsistency, you can't be sure that `validatorRewardAddressToId` really maps rewardAddress to id 2. `getNodeOperator(rewardAddress)` will return wrong value for this address 3. Overall misleading and prone to error

**Recommendation**

Consider clearing `validatorRewardAddressToId` on `setRewardAddress`

**Update**

*Fixed* as recommended.
2.4.75 Copying storage validatorIds to memory has no point

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#L377

```solidity
uint256[] memory memValidatorIds = validatorIds;
```

In our calculations of gas costs it costs more to copy than to just use the storage array for your use cases.

Recommendation

Don't copy validatorIds to memory, use it as is.

Update

Fixed 1, 2, 3 as recommended.

2.4.76 Term 'validator' has different meanings throughout the codebase

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>ACKNOWLEDGED</td>
</tr>
</tbody>
</table>

Description

It mean both IStakeManager.Validator and ValidatorData in this example (activeValidators and validator variables). NodeOperatorRegistry.sol#L379-L387

```solidity
IStakeManager.Validator memory validator;
NodeOperatorRegistryStatus operatorStatus;
ValidatorData[]
    memory activeValidators = new ValidatorData[](length);

for (uint256 i = 0; i < length; i++) {
    (operatorStatus, validator) = _getOperatorStatusAndValidator(
        memValidatorIds[i]
    );
```
Recommendation

Consider creating a terminology that you will use throughout the codebase. So validator and other terms mean only one thing. E.g. smValidator, validatorAddresses.

2.4.77 Unnecessary nesting

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#L388-L399

```solidity
if (operatorStatus == NodeOperatorRegistryStatus.ACTIVE) {
    if (!IValidatorShare(validator.contractAddress).delegation())
        continue;

    activeValidators[
        totalActiveNodeOperators
    ] = ValidatorData(
        validator.contractAddress,
        validatorIdToRewardAddress[memValidatorIds[i]]
    );
    totalActiveNodeOperators++;
}
```

May be replaced with early 'continue'.

Recommendation

Consider replacing with

```solidity
if (operatorStatus != NodeOperatorRegistryStatus.ACTIVE) {
    continue;
}
if (!IValidatorShare(validator.contractAddress).delegation()) {
    continue;
}
activeValidators[
    totalActiveNodeOperators
] = ValidatorData(
    validator.contractAddress,
    validatorIdToRewardAddress[memValidatorIds[i]]
);```
2.4.78 totalValidatorToWithdrawFrom formula does not follow the docs

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

In docs, you wrote
\[
\text{numValidators} = \text{Min}((\text{requestAmountPercentage} + \text{minRequestWithdrawRange}) / \text{delegationPercentagePerValidator}), \text{validatorDelegatedAmount.length})
\]

In the code, \( (\text{requestAmountPercentage} + \text{minRequestWithdrawRange}) / \text{delegationPercentagePerValidator} \) is replaced with \( (\text{requestAmountPercentage} + \text{minRequestWithdrawRange}) / \text{delegationPercentagePerValidator} + 1 \).

NodeOperatorRegistry.sol#L771-L774

\[
\text{totalValidatorToWithdrawFrom} = ((\text{withdrawAmountPercentage} + \text{MIN_REQUEST_WITHDRAW_RANGE}) / (100 / \text{length})) + 1;
\]

**Recommendation**

Update the docs.

**Update**

Fixed in the docs.

2.4.79 Mutating a variable instead of using several

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

**Description**

When you use one variable to calculate a value it's hard to follow. For example, NodeOperatorRegistry.sol#L800-L806
\[\text{uint256 rebalanceTarget} = \text{totalDelegated} > \_\text{withdrawAmount} \]
\[\quad \text{(totalDelegated - \_\text{withdrawAmount})} / \text{length} \]
\[\quad \colon 0;\]

\[\text{rebalanceTarget} = \text{rebalanceTarget} > \text{minAmount} \]
\[\quad \text{? minAmount} \]
\[\quad \colon \text{rebalanceTarget};\]

Recommendation

Consider using several variables or a function to calculate the final value.

### 2.4.80 Possibly undesired withdrawal proportions

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>ACKNOWLEDGED</td>
</tr>
</tbody>
</table>

**Description**

If there is a new validator with 0 stake (or one with a small stake) rebalanceTarget is always 0 (or \(~0, = \text{minAmount}\) NodeOperatorRegistry.sol\#L804-L806

\[\text{rebalanceTarget} = \text{rebalanceTarget} > \text{minAmount} \]
\[\quad \text{? minAmount} \]
\[\quad \colon \text{rebalanceTarget};\]

operatorAmountCanBeRequested[idx] will be 100% of validator's stake (or \(~100\%) NodeOperatorRegistry.sol\#L821-L825

\[\text{uint256 operatorRatioToRebalance} = \text{stakePerOperator}[idx] \neq 0 \&\& \]
\[\quad \text{stakePerOperator}[idx] - \text{rebalanceTarget} > 0 \]
\[\quad \text{? stakePerOperator}[idx] - \text{rebalanceTarget} \]
\[\quad \colon 0;\]
\[\text{operatorAmountCanBeRequested}[idx] = \text{operatorRatioToRebalance};\]

So we will withdraw everything from the first big one, then from the next big one, etc Which may not be desired because it withdraws to less than average.

**Recommendation**

Consider rewriting logic for requesting withdrawal amounts.
2.4.81 Dangerous calculation

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#L822

```solidity
stakePerOperator[idx] - rebalanceTarget > 0
```

It does not underflow here but it may be replaced with a safer one to avoid errors in the future changes.

Recommendation

Replace with `stakePerOperator[idx] > rebalanceTarget`

Update

Fixed as recommended.

2.4.82 Reusing a variable in for-loop reduce readability

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

Description

NodeOperatorRegistry.sol#L565

```solidity
uint256 operatorRatioToDelegate;

for (uint256 idx = 0; idx < totalActiveNodeOperator; idx++) {
    operatorRatioToDelegate = stakePerOperator[idx] >= rebalanceTarget
    ...
}
```

It’s best to declare a variable in a smallest scope possible. Inside for loop in this case. Furthermore it costs more gas to keep a variable outside if you compile with enabled optimizations.
Recommendation

Declare `operatorRatioToDelegate` inside for-loop.

2.4.83 `DISTANCE_THRESHOLD` read several times from storage

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

In `getValidatorsDelegationAmount` first here `NodeOperatorRegistry.sol#L551`

```solidity
if (distanceThreshold <= DISTANCE_THRESHOLD) {
```

And then several times in for-loop `NodeOperatorRegistry.sol#L575`

```solidity
for (uint256 idx = 0; idx < totalActiveNodeOperator; idx++) {
    operatorRatioToDelegate = stakePerOperator[idx] >= rebalanceTarget
        ? 0
        : rebalanceTarget - stakePerOperator[idx];

    if (operatorRatioToDelegate != 0 && stakePerOperator[idx] != 0) {
        operatorRatioToDelegate = (rebalanceTarget * 100) / stakePerOperator[idx] >= DISTANCE_THRESHOLD
            ? operatorRatioToDelegate
            : 0;
    }

    operatorRatios[idx] = operatorRatioToDelegate;
    totalRatio += operatorRatioToDelegate;
}
```

Recommendation

Consider copying to memory to save gas.

Update

Fixed as recommended.
2.4.84 Issues from report for PR#69 are not fixed here

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

Issues from report 69 are not addressed in this commits

**Recommendation**

Fix in future commits

**Update**

Fixed in current version as written in the report.

2.4.85 Redundant check

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>NO_ISSUE</td>
</tr>
</tbody>
</table>

**Description**

PoLidoNFT.sol#L130

```solidity
if (burnedTokenIndexInOwnerTokens != lastOwnerTokensIndex &&
    ownerTokensLength != 1)
{

ownerTokensLength != 1 check is redundant because when
ownerTokensLength == 1 burnedTokenIndexInOwnerTokens ==
lastOwnerTokensIndex. The reason is that there is only one token, so it's both last and burned.
```

The same is valid for PoLidoNFT.sol#L223

```solidity
if (removeApprovedTokenIndexInOwnerTokens != lastApprovedTokensIndex
&&
    approvedTokensLength != 1)
{
```
Recommendation
Remove redundant check

2.4.86 Abstruse code

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description
PoLidoNFT.sol#L131-L137

This part of code is hard to follow. The same is valid for _removeApproval
PoLidoNFT.sol#L225-L233

Recommendation
Consider adding comments or extracting a function, e.g.

```solidity
/* Swap burned token with the last one */

uint256 lastOwnerTokenId = ownerTokens[lastOwnerTokensIndex];
// Make the last token have an index of a token we want to burn.
// So when we request index of token with id that is currently last in ownerTokens it does not point
// to the last slot in ownerTokens, but to a burned token's slot (we
will update the slot at the next line)
token2Index[lastOwnerTokenId] = burnedTokenIndexInOwnerTokens;
// Copy currently last token to the place of a token we want to burn.
// So updated pointer in token2Index points to a slot with the
correct value.
ownerTokens[burnedTokenIndexInOwnerTokens] = ownerTokens[
    lastOwnerTokensIndex
];

Update

**Fixed** as recommended.

### 2.4.87 owner2Tokens[from]'s length does not decrease on a transfer

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

When a token is burned the length of the array owner2Tokens[from] decreases.

But not on transfer. It may be confusing to 3rd parties and in future development.

**PoLidoNFT.sol#L146-L159**

**Recommendation**

Consider removing a transferred token on transfer as you do on burn.

**Update**

**Fixed** as recommended.

### 2.4.88 Forgotten import "hardhat/console.sol";

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

**StMATIC.sol#L17**
import "hardhat/console.sol";

Recommendation
Remove unused import.

Update
Fixed as recommended.

2.4.89 Typo

Severity | INFO  
--|---
Status  | FIXED

Description
StMATIC.sol#L87

```sol
/// @notice When an operator quite the system StMATIC contract
withdraw the total delegated
```

Recommendation
Replace quite with quit.

Update
Fixed as recommended.

2.4.90 Singular noun is used for an array

Severity | INFO  
--|---
Status  | FIXED

Description
StMATIC.sol#L89

```sol
RequestWithdraw[] public stMaticWithdrawRequest;
```

Recommendation
Rename to stMaticWithdrawRequests
Update

**Fixed** as recommended.

### 2.4.91 Redundant array copy

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

StMATIC.sol#L670

```solidity
RequestWithdraw[]
  memory stMaticWithdrawRequestCache = stMaticWithdrawRequest;
```

There is no need for cache, it only cost additional gas because you read from each storage slot once anyway. And you pay for a creation and filling of the memory array.

**Recommendation**

Remove redundant array copy

**Update**

**Fixed** as recommended.

### 2.4.92 Plural noun is used for singular object

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

**Description**

StMATIC.sol#L690

```solidity
RequestWithdraw memory lidoRequests = stMaticWithdrawRequest[_index];
```

**Recommendation**

Rename to `lidoRequest`

**Update**

**Fixed** as recommended.
2.4.93 Misleading function name

<table>
<thead>
<tr>
<th>Severity</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>FIXED</td>
</tr>
</tbody>
</table>

Description

/StMATIC.sol#L1119

```solidity
function _getMaticFromTokenId(RequestWithdraw memory requestData) {

    // The function name says FromTokenId but uses requestData

    // Recommendation
    Rename the function to _getMaticFromRequestData

    // Update
    Fixed as recommended.
```
3 Conclusion

The following table contains the total number of issues that were found during audit:

<table>
<thead>
<tr>
<th>Level</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRITICAL</td>
<td>0</td>
</tr>
<tr>
<td>MAJOR</td>
<td>0</td>
</tr>
<tr>
<td>WARNING</td>
<td>14</td>
</tr>
<tr>
<td>INFO</td>
<td>93</td>
</tr>
<tr>
<td>Total</td>
<td>107</td>
</tr>
</tbody>
</table>

Smart contracts have been audited and no critical or major issues were found. Also a lot of recommendations were marked as warning and informational. Some changes were proposed to follow best practices, reduce potential attack surface, simplify code maintenance and increase its readability. As stated in each particular issue, all issues identified have been correctly fixed, acknowledged or marked as "no issue" after a discussion with the client. Contracts are assumed as secure to use according to our security criteria and ready to deploy to mainnet.
About Oxorio

Oxorio is a young but rapidly growing audit and consulting company in the field of the blockchain industry, providing consulting and security audits for organizations from all over the world. Oxorio has participated in multiple blockchain projects where smart contract systems were designed and deployed by the company.

Oxorio is the creator, maintainer, and major contributor of several blockchain projects and employs more than 5 blockchain specialists to analyze and develop smart contracts.

Contacts:

- oxor.io
- ping@oxor.io
- github
- linkedin