

STATE MIND

Lido on Polygon v2

20-03-2023 – 17-05-2023

Table of contents



1. Project Brief		3
2. Finding Severity breakdown		4
3. Summary of findings		5
4. Conclusion		5
5. Findings report		7
High	Incorrect minimum calculation	7
Medium	Broken delegated flag	8
	Excessive return data	8
	Insufficient input validation	9
	Wrong calculation of the minAmount variable	9
	Incorrect delegate allocation	10

Informational	Optimize mint function	10
	Bad comments	11
	Differences in naming of Matic	11
	Zero-valued activeValidators during requestWithdraw	12
	Excessive memory usage	13
	Gas optimization via trimming memory arrays	13
	Redundant SLOADs	14
	Excessive SLOAD	15
	Variable mismatching	15
	Code duplication for a minimum of two values	15
	Zero check	16
	Extra calculations	16
	Storage clearing	16
	Wrong return value	16
	Gas optimization	17
	Code duplication	17
	The minAmount variable can be bigger than the actual amount	18
	Unclear naming	18
	Better naming of global variables	18
	Redundant inheritance	19

6. Appendix A. Linter	20
-----------------------	----

7. Appendix B. Slither	22
------------------------	----

8. Appendix C. Tests	27
----------------------	----

1. Project Brief



Title	Description
Client	Lido
Project name	Lido on Polygon v2
Timeline	20-03-2023 - 17-05-2023
Initial commit	da2e7ee19ab8552ed278f9e3a3a25be57b8068cc
Final commit	eb197c6c189a07c1f30b72206ccb41ae02b639fc

Short Overview

Overview


The LIDO contracts on the Polygon blockchain are for liquid staking. Because the Polygon's consensus layer works in the Ethereum network, the central LIDO contracts are located in it too.

Users can stake their ERC20 Matic tokens in the Ethereum network to receive stMatic tokens.

Project Scope

The audit covered the following files:

 [StMATIC.sol](#)

 [NodeOperatorRegistry.sol](#)

 [PoLidoNFT.sol](#)

 [RateProvider.sol](#)

 [FxStateRootTunnel.sol](#)

 [FxStateChildTunnel.sol](#)

 [FxBaseRootTunnel.sol](#)

 [FxBaseChildTunnel.sol](#)

 [ExitPayloadReader.sol](#)

 [Merkle.sol](#)

 [MerklePatriciaProof.sol](#)

 [RLPReader.sol](#)

2. Finding Severity breakdown



All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds to be transferred to any party.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss of funds.
Informational	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

3. Summary of findings



Severity	# of Findings
Critical	0 (0 fixed, 0 acknowledged)
High	1 (1 fixed, 0 acknowledged)
Medium	5 (5 fixed, 0 acknowledged)
Informational	21 (19 fixed, 2 acknowledged)
Total	27 (25 fixed, 2 acknowledged)

4. Conclusion



Deployment

File name	Contract deployed on mainnet
PoLidoNFT (Proxy)	<u>0x60a91E2B7A1568f0848f3D43353C453730082E46</u>
PoLidoNFT (Implementation)	<u>0x41912d95d040ecc7d715e5115173d37e4e7cb24e</u>
StMATIC (Proxy)	<u>0x9ee91F9f426fA633d227f7a9b000E28b9dfd8599</u>
StMATIC (Implementation)	<u>0x6c25aebd494a9984a3d7c8cf395c8713e0c74d98</u>
NodeOperatorRegistry (Proxy)	<u>0x216B8b78e0632138dc38907dd089aAB601ED6EDC</u>
NodeOperatorRegistry (Implementation)	<u>0xdafa0332e904d5d1f2f1cdc2a3cad636a120f72d</u>
FxStateRootTunnel	<u>0xc7dd5c30DcA04f487c9ede0c5AC580c91587fc66</u>
FxStateChildTunnel	<u>0x0833f5bD45803E05ef54E119a77E463cE6b1a963</u>

File name	Contract deployed on mainnet
RateProvider	<u>0xdEd6C522d803E35f65318a9a4d7333a22d582199</u>

5. Findings report



HIGH-01

Incorrect minimum calculation

Fixed at [24a0e6](#)

Description

The **NodeOperatorRegistry** contract has several places, where the minimum value **minAmount** might be calculated wrong. If an input **array** has **0** value with an **index** where **index < array.length - 1**, then the algorithm essentially would return a minimum of **array[index + 1:array.length]**.

```
if (minAmount > amount || minAmount == 0) {  
    minAmount = amount;  
}
```

E.g. [20, 33, 0, 100, 51] -> 51

- 1. [_getValidatorsDelegationInfos](#)
- 2. [_getValidatorsRequestWithdraw](#)
- 3. [getProtocolStats](#)

Those functions are part of the main protocol flow (deposits & withdrawals) and could lead to significant calculation errors.

Recommendation

We recommend setting **minAmount = type(uint256).max** before the loop and checking if each value is less than **minAmount**.

MEDIUM-01	Broken delegated flag	Fixed at e47dd3
-----------	-----------------------	---------------------------------

Description

if function we check if system balanced or not code but in other function we calculate it's as

```
uint256 distanceThresholdPercents = DISTANCE_THRESHOLD_PERCENTS;
bool isTheSystemBalanced = distanceThreshold <=
    distanceThresholdPercents;
```

In result in some configurations **getValidatorsRebalanceAmount** and **getValidatorsDelegationAmount** would recognize both Balanced and Not balanced. Example:

```
number = [1, 2, 3]
totalStaked = [120, 100, 113]
```

As an example of rebalance when it's not needed

```
number = [1, 2, 3, 4]
totalStaked = [12, 10, 10, 10]
```

rebalance target would be 10 and we withdraw from 1 operator 2 matic when system is balanced.

Recommendation

We recommend changing this require to

```
require(
    distanceThreshold > distanceThresholdPercents && totalStaked > 0,
    "The system is balanced"
);
```

MEDIUM-02	Excessive return data	Fixed at eee9d6
-----------	-----------------------	---------------------------------

Description

NodeOperatorRegistry::_getValidatorsRequestWithdraw should return only non **Inactive** validators. At the same time, there is no check for status value during a loop combining return data

Recommendation

We recommend skipping inactive validators during the loop:

```
(validatorStatus, validator) = _getOperatorStatusAndValidator(validatorId);

if (validatorStatus == NodeOperatorRegistryStatus.INACTIVE)
    continue;
```

MEDIUM-03	Insufficient input validation	Fixed at 534556
-----------	-------------------------------	---------------------------------

Description

Validator could brake contract logic by passing his own address to the **NodeOperatorRegistry**'s setRewardAddress function.

Recommendation

We recommend adding a new input check:

```
require(_newRewardAddress != msg.sender, "Invalid reward address");
```

Or changing the order of mapping modifications:

```
delete validatorRewardAddressTold[msg.sender]; // move up
validatorIdToRewardAddress[validatorId] = _newRewardAddress;
validatorRewardAddressTold[_newRewardAddress] = validatorId;
```

MEDIUM-04	Wrong calculation of the minAmount variable	Fixed at 5efe1d
-----------	---------------------------------------------	---------------------------------

Description

In the contract **NodeOperatorRegistry** in the function _getValidatorsRequestWithdraw, the contract sometimes miscalculates the variable **minAmount**.

The problem is in the if-clauses here. For example, in case there are **[0, 1000]** amounts of validators, the return parameters **minAmount** and **maxAmount** will be equal to **1000**. But the problem with this wrong calculation occurs in the getValidatorsRequestWithdraw function in cases when **totalDelegated != 0**. There are two possible scenarios: return here and at the end of the function.

In the first scenario, the contract will count that the system is balanced when it is not, and the first **totalValidatorToWithdrawFrom** validators have at least **min(_withdrawAmount, totalDelegated) / totalValidatorToWithdrawFrom** amounts. In the case above, when validators' amounts are **[0, 1000]** and a user wants to withdraw the amount in the StMatic token equivalent to **100** Matic tokens, the contract will assume that the first validator will have at least **100** Matic tokens, which is wrong. This wrong calculation will lead to an error in the **StMATIC** contract in the function requestWithdraw. The contract will call the _requestWithdrawBalanced function because it assumes the system is balanced. This assumption will lead to a revert of a transaction (calling in the **validatorShare** contract the function **sellVoucher_new**) because the contract will ask a validator to withdraw tokens that it doesn't have.

In the second scenario, there will be no significant errors in the following calculations and actions because of this wrong value.

Recommendation

In the contract **NodeOperatorRegistry** in the function _getValidatorsRequestWithdraw, in if-clause remove condition **&& amount != 0**. In order not to break if-statement at Lines 773 - 776, because there could be division by zero, it is recommended to change the order of conditions or during division check if **minAmount** is zero.

MEDIUM-05	Incorrect delegate allocation	Fixed at 2f2196
-----------	-------------------------------	---------------------------------

Description

In the function `delegate` the condition `if (totalRatio == 0)` is used to check that the system is in a balanced state. However, this is an incorrect check and there may be cases where the system is unbalanced but tokens will be delegated as the system is balanced. Here is an example of such a case.

```
distanceThresholdPercents = 120
stakePerOperator = [1000, 1000, 1000, 1300]
totalDelegated = 1000 + 1000 + 1000 + 1300 = 4300
isTheSystemBalanced = False
amountToDelegate = 100
rebalanceTarget = (4300 + 100) / 4 = 1100
operatorRatioToDelegate = [0, 0, 0, 0]
totalRatio = 0
```

Recommendation

We recommend adding variable `isTheSystemBalanced` to return for function `getValidatorsDelegationAmount` and use it in `delegate`.

Client's comments

```
It's a known issue and is included in a fix - https://github.com/lidofinance/polygon-contracts/blob/main/audits/v2/Shardlabs/LoP-v2-delegation.pdf
```

INFORMATIONAL-01	Optimize mint function	Fixed at dd232f
------------------	------------------------	---------------------------------

Description

In this `function` variable `currentIndex` is created. But it possible to only change `tokenIdIndex` once.
 UPDATE: This version is better readable, but previous implementation was consuming less gas. If gas usage is important, it is better to leave previous version.

Recommendation

We recommend rewriting `mint` to

```
function mint(address _to) external override isLido returns (uint256) {
    _mint(_to, ++tokenIdIndex);
    return tokenIdIndex;
}
```

INFORMATIONAL-02	Bad comments	Fixed at c868eb
<p>Description</p> <p><u>1</u> minAmount the distance between the min and max amount staked in a validator <u>2</u> maxAmount the distance between the min and max amount staked in a validator Same comments for different varibales.</p> <p>Recommendation</p> <p>We recommend changing it:</p> <p>1 - minimum amount staked in a validator 2 - maximum amount staked in a validator</p>		

INFORMATIONAL-03	Differences in naming of Matic	Fixed at b33f19
<p>Description</p> <p><u>there</u> is two variables stMATIC, MATIC, but in RateProvider.sol they named uint256 stMatic, uint256 matic</p> <p>Recommendation</p> <p>We recommend to unify those variables (better with second variant)</p>		

INFORMATIONAL-04	Zero-valued activeValidators during requestWithdraw	Fixed at b3c4b6
------------------	-----------------------------------------------------	---------------------------------

Description

The `getValidatorsRequestWithdraw` function gets a list of all active validators via the internal `_getValidatorsRequestWithdraw` call. `_getValidatorsRequestWithdraw` iterates over all validators and skips invalid ones. Since the list is not altered even if we skip during a `for` loop, we would have inside it a zero-valued `struct ValidatorData`. E.g.: `[ACTIVE, INACTIVE, ACTIVE, ACTIVE] -> [{shareAddr, rewardAddr}, {0, 0}, {shareAddr, rewardAddr}, {shareAddr, rewardAddr}]`

Which, if stakes are balanced, ultimately would lead to reverting during `_requestWithdrawBalanced` in an attempt to call zero-address (`_calculateValidatorShares`).

```
uint256 exchangeRatePrecision = _getExchangeRatePrecision(
    validatorShare.validatorId() // <- will revert
);
uint256 rate = validatorShare.exchangeRate();
```

Furthermore, the `length` value would be incorrect, which affects `balanced` or `unbalanced` calculations.

```
(
    validators,
    stakePerOperator,
    totalDelegated,
    minAmount,
    maxAmount
) = _getValidatorsRequestWithdraw();

...

uint256 length = validators.length; // <-- The length value is more than it has to be.
uint256 withdrawAmountPercentage = (_withdrawAmount * 100) /
    totalDelegated;

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

totalValidatorToWithdrawFrom = min(totalValidatorToWithdrawFrom, length);

if (
    (maxAmount * 100) / minAmount <= DISTANCE_THRESHOLD_PERCENTS &&
    minAmount * totalValidatorToWithdrawFrom >= _withdrawAmount
){
    ...
```

Recommendation

We recommend using a new counter variable to populate the `activeValidators` list as well as trimming it afterward.

INFORMATIONAL-05	Excessive memory usage	Fixed at <u>9b75be</u>
------------------	------------------------	------------------------

Description

In the `getValidatorsDelegationAmount` function, `DISTANCE_THRESHOLD_PERCENTS` storage variable is saved into the memory variable and used only one time.

The same optimization could be applied in the `getProtocolStats` and `getStats` functions with `validatorIdToRewardAddress[validatorId]` variable.

Recommendation

We recommend using storage value if saving into memory is unnecessary.

INFORMATIONAL-06	Gas optimization via trimming memory arrays	Fixed at <u>613926</u>
------------------	---------------------------------------------	------------------------

Description

If a return array has more allocated size than it uses, then we can reduce gas consumption by trimming such an array.

- `listDelegatedNodeOperators::activeValidators`
- `listWithdrawNodeOperators::withdrawValidators`
- `_getValidatorsDelegationInfos::validators` and `_getValidatorsDelegationInfos::stakePerOperator`

UPDATE: Trimming could be optimized further by removing redundant operations. New length's value could be set as `realLength` value. Also arrays `bigNodeOperatorIds` and `smallNodeOperatorIds` should be trimmed.

```

if (realLength < fullLength) {
  assembly {
    mstore(array, realLength)
  }
}
```

Recommendation

We recommend using the same trimming technique as [Lido on Ethereum](#).

```

if (realLength < fullLength) {
  uint256 trim = fullLength - realLength;
  assembly {
    mstore(array, sub(mload(array), trim))
  }
}
```

INFORMATIONAL-07	Redundant SLOADs	Fixed at 5b4747
------------------	------------------	---------------------------------

Description

The internal `_removeApproval` function reads `_tokenApprovals` mapping to get an approved address using `getApproved`. At the same time, every `_removeApproval` is called inside an if statement `getApproved(_tokenId) != address(0)`. Essentially, if `getApproved(_tokenId) != address(0)` then we would make two **SLOAD** operations consecutively for the same slot.

```
if (getApproved(tokenId) != address(0)) {
    _removeApproval(tokenId);
}
```

```
function _removeApproval(uint256 _tokenId) internal {
    uint256[] storage approvedTokens = address2Approved[
        getApproved(_tokenId)
    ];
```

1. `approve`
2. `_beforeTokenTransfer::Burning`
3. `_beforeTokenTransfer::Transferring`

The same approach could be used to optimize `_getOperatorStatusAndValidator(uint256 _validatorId)` with **SLOAD** of `validatorIdToRewardAddress[_validatorId]`.

1. `removeInvalidNodeOperator`
2. `listDelegatedNodeOperators`
3. `listWithdrawNodeOperators`
4. `_getValidatorsDelegationInfos`
5. `_getValidatorsRequestWithdraw`
6. `getNodeOperator`

UPDATE:

The `getNodeOperator` function reads the storage variable `validatorIdToRewardAddress[_validatorId]` twice.

```
(
    NodeOperatorRegistryStatus operatorStatus,
    IStakeManager.Validator memory validator
) = _getOperatorStatusAndValidator(
    _validatorId,
    validatorIdToRewardAddress[_validatorId] // <-- SLOAD
);
nodeOperator.validatorShare = validator.contractAddress;
nodeOperator.validatorId = _validatorId;
nodeOperator.rewardAddress = validatorIdToRewardAddress[_validatorId]; // <-- SLOAD
...
```

Recommendation

We recommend saving the result of the `getApproved` call to memory and passing it to the `_removeApproval` function.

```
address approvedAddress = getApproved(tokenId);
if (approvedAddress != address(0)) {
    _removeApproval(tokenId, approvedAddress);
}
```

INFORMATIONAL-08	Excessive SLOAD	Fixed at c1f3f9
------------------	-----------------	---------------------------------

Description

The internal `_beforeTokenTransfer` function saves `ownerTokens[lastOwnerTokensIndex]` into the memory as `lastOwnerTokenId`, but reads the same slot while saving `ownerTokens[burnedTokenIndexInOwnerTokens]`.

```
uint256 lastOwnerTokenId = ownerTokens[lastOwnerTokensIndex];

token2Index[lastOwnerTokenId] = burnedTokenIndexInOwnerTokens;

ownerTokens[burnedTokenIndexInOwnerTokens] = ownerTokens[lastOwnerTokensIndex]; // same as lastOwnerTokenId
```

Recommendation

We recommend reusing the memory variable.

```
uint256 lastOwnerTokenId = ownerTokens[lastOwnerTokensIndex];

token2Index[lastOwnerTokenId] = burnedTokenIndexInOwnerTokens;

ownerTokens[burnedTokenIndexInOwnerTokens] = lastOwnerTokenId;
```

INFORMATIONAL-09	Variable mismatching	Fixed at a53eb9
------------------	----------------------	---------------------------------

Description

There is a couple of instances where an expression used 2 variables with the same value, but different name `amountCanBeRequested = allowedAmountToRequestFromOperators[id]`;

```
amount2WithdrawFromValidator = amountCanBeRequested > currentAmount2WithdrawInMatic
    ? currentAmount2WithdrawInMatic
    : allowedAmountToRequestFromOperators[id];
```

Recommendation

We recommend using one variable to denote one value

INFORMATIONAL-10	Code duplication for a minimum of two values	Fixed at 29819e
------------------	----------------------------------------------	---------------------------------

Description

The protocol calculates a minimum of two values using a ternary operator, which could be replaced with a simple function call.

```
totalValidatorToWithdrawFrom = totalValidatorToWithdrawFrom > length
    ? length
    : totalValidatorToWithdrawFrom;
```

[StMATIC.sol#L305](#) [StMATIC.sol#L348](#)
[NodeOperatorRegistry.sol#L751](#) [NodeOperatorRegistry.sol#L779](#)

Recommendation

We recommend introducing a new internal `min(uint256, uint256)` function to improve code quality and readability.

INFORMATIONAL-11	Zero check	Fixed at a02118
<p>Description</p> <p>At Lines 610 - 614 operatorRatioToRebalance is calculated, but if operatorRatioToRebalance is zero there is no sense in calculations.</p> <p>Recommendation</p> <p>It is recommended to add zero check for operatorRatioToRebalance variable before calculation.</p>		
INFORMATIONAL-12	Calculation of an unnecessary variable	Fixed at e06127
<p>Description</p> <p>At Lines 772 - 774 withdrawAmountPercentage is recalculated, but then it is never used.</p> <p>Recommendation</p> <p>It is recommended to leave same value for withdrawAmountPercentage variable.</p>		
INFORMATIONAL-13	Extra calculations	Fixed at 203951
<p>Description</p> <p>At Lines 434 - 437 in case of tatolRatio equals 0, variable amountToDelegatePerOperator is calculated every iteration, but it can be calculated once.</p> <p>Recommendation</p> <p>It is recommended to calculate once variable amountToDelegatePerOperator before cycle if case of totalRatio equals zero.</p>		
INFORMATIONAL-14	Storage clearing	Fixed at 2e4571
<p>Description</p> <p>At Line 492 poLidoNFT is burned, but token2WithdrawRequests[_tokenId] is still in storage.</p> <p>Recommendation</p> <p>It is recommended to clear storage after burning NFT and claiming tokens.</p>		
INFORMATIONAL-15	Wrong return value	Fixed at 1a5505
<p>Description</p> <p>In the contract NodeOperatorRegistry in the function getProtocolStats in some cases there can be a wrong return value minAmount. In case there are no validators return value minAmount will be equal to the type(uint256).max value.</p> <p>Recommendation</p> <p>It is recommended to handle this case properly.</p>		

INFORMATIONAL-16	Gas optimization	Fixed at 4fa8a3
------------------	------------------	---------------------------------

Description

In this contracts there are some non-optimal work with global variables that leads to waste of gas:

- NodeOperatorRegistry.
 - In the function `_removeOperator` on the [L201](#) and [L204](#) the global variable is read multiple times
 - In the function `listDelegatedNodeOperators` on the [L353](#) the global variable `validatorIds.length` is read multiple times. Also, the global variable `validatorIds[i]` is read multiple times too
 - In the function `_getValidatorsDelegationInfos` on the [L449](#) the global variable `stMATIC` is read multiple times. Also, the global variable `validatorIds[i]` is read multiple times too. The same in the function `_getValidatorsRequestWithdraw`
- PoLidoNFT
 - In the function `_beforeTokenTransfer` the global variable `ownerTokens[lastOwnerTokensIndex]` is read multiple times ([here](#) and [here](#))
- StMATIC
 - In the functions `delegate`, `_claimTokensV2`, `_claimTokensV1`, `distributeRewards`, and `claimTokensFromValidatorToContract` the global variable `token` is read multiple times
 - In the function `requestWithdraw` global variables `totalBuffered` and `reservedFunds` are read multiple times

UPDATE:

In this contracts there left some non-optimal work with global variables that leads to waste of gas:

- NodeOperatorRegistry.
 - In the function `_getValidatorsDelegationInfos` on the [L467](#) the global variable `stMATIC` is read multiple times.
 - In the function `_getValidatorsRequestWithdraw` on the [L680](#) the global variable `stMATIC` is read multiple times. Also, the global variable `validatorIds[i]` is read multiple times too ([L672](#) and [L673](#)).
- StMATIC
 - In the function `distributeRewards` the global variable `token` is read multiple times
 - In the function `delegate` the global variable `stakeManager` is read ([here](#) and [here](#)) twice
 - In the function `_createWithdrawRequest` the global variable `stakeManager` is read multiple times

Recommendation

It is recommended to use variables in the stack or in the memory to optimize the work with global variables.

INFORMATIONAL-17	Code duplication	Fixed at 7bcf10
------------------	------------------	---------------------------------

Description

This contracts have parts in their's code that are similar and can be replaced with one helper function.

- NodeOperatorRegistry.
 - The functions `listDelegatedNodeOperators` and `listWithdrawNodeOperators`
 - The functions `getNodeOperator` ([this](#) and [this](#))

Recommendation

It is recommended to add helper functions that will replace similar parts of codes. This will reduce code repeating.

INFORMATIONAL-18	The minAmount variable can be bigger than the actual amount	Fixed at 058edd
<p>Description</p> <p>In the contract NodeOperatorRegistry in the function _getValidatorsRequestWithdraw on the L677 there is a conversion of the minAmount variable. This variable can be equal to 1 even the minimum amount is 0. The problem is that this value is passed to the getValidatorsRequestWithdraw function and is used on the L757. On that line the contract checks if the contract StMATIC can withdraw _withdrawAmount amount of tokens from the first totalValidatorToWithdrawFrom validators. The problem is that the contract may tell that minAmount * totalValidatorToWithdrawFrom >= _withdrawAmount, but the contract won't be able to withdraw _withdrawAmount amount of tokens, because some validator may have zero balance.</p> <p>Recommendation</p> <p>It is recommended to remove the <u>conversion</u> of the minAmount variable in the _getValidatorsRequestWithdraw function and use it on the L756 for calculating distanceThreshold.</p>		

INFORMATIONAL-19	Unclear naming	Fixed at 733dc0
<p>Description</p> <p>There are several strange naming for variables</p> <ul style="list-style-type: none"> • distanceThreshold actually not a threshold and can be confused with distanceThresholdPercents • operatorRatios actually not ratios and can be incorrectly treated as ratios. • totalValidatorToWithdrawFrom consider totalValidatorsToWithdrawFrom • nonInactiveValidators consider activeValidators <p>Recommendation</p> <p>We recommend making clear naming for the described variables.</p>		

INFORMATIONAL-20	Better naming of global variables	Acknowledged
<p>Description</p> <p>1, 2, 3</p> <p>All of that - global variables, not constants.</p> <p>In third case variable each time cast to uint8, it's more expensive.</p> <p>Recommendation</p> <p>We recommend to rename variables to distance_threshold_percents, max_withdraw_percentage_per_rebalance, min_request_withdraw_range_percents and setting 3 to uint256 instead of uint8. (Little gas optimisation)</p>		

Description

The **PoLidoNFT** contract inherits **ERC721Upgradeable** and **ERC721PausableUpgradeable** while **ERC721PausableUpgradeable** inherits **ERC721Upgradeable**.

Recommendation

We recommend removing **ERC721Upgradeable** inheritance from the **PoLidoNFT** contract.

Client's comments

As the contract is already deployed, and to not risk modifying the storage we decided to not fix this issue.

6. Appendix A. Linter



Error/ordering

- interfaces/INodeOperatorRegistry.sol:252 – Function order is incorrect, event definition can not go after external view function (line 232)
- interfaces/IPoLidoNFT.sol#L31 – Function order is incorrect, external function can not go after external view function (line 24)
- interfaces/IStakeManager.sol:42 – Function order is incorrect, external function can not go after external view function (line 36)
- interfaces/IStMATIC.sol:129 – Function order is incorrect, external function can not go after external view function (line 120)
- interfaces/IValidatorShare.sol:22 – Function order is incorrect, external function can not go after external view function (line 20)
- NodeOperatorRegistry.sol:223 – Function order is incorrect, external function can not go after private function (line 196)

Warning/no-global-import

- interfaces/IStMATIC.sol:7 – global import of path `./IValidatorShare.sol` is not allowed. Specify names to import individually or bind all exports of the module into a name (`import "path" as Name`)
- interfaces/IStMATIC.sol:8 – global import of path `./INodeOperatorRegistry.sol` is not allowed. Specify names to import individually or bind all exports of the module into a name (`import "path" as Name`)
- `[interfaces/IStMATIC.sol:9](https://github.com/lidofinance/polygon-contracts/blob/da2e7ee19ab8552ed278f9e3a3a25be57b8068cc/contracts/interfaces/IStMATIC.sol#L9)` – global import of path `./IStakeManager.sol` is not allowed. Specify names to import individually or bind all exports of the module into a name (`import "path" as Name`)
- `[interfaces/IStMATIC.sol:10](https://github.com/lidofinance/polygon-contracts/blob/da2e7ee19ab8552ed278f9e3a3a25be57b8068cc/contracts/interfaces/IStMATIC.sol#L10)` – global import of path `./IPoLidoNFT.sol` is not allowed. Specify names to import individually or bind all exports of the module into a name (`import "path" as Name`)
- interfaces/IStMATIC.sol:11 – global import of path `./IFxStateRootTunnel.sol` is not allowed. Specify names to import individually or bind all exports of the module into a name (`import "path" as Name`)
- NodeOperatorRegistry.sol:8 – global import of path `./interfaces/IValidatorShare.sol` is not allowed. Specify names to import individually or bind all exports of the module into a name (`import "path" as Name`)
- `[NodeOperatorRegistry.sol:9](https://github.com/lidofinance/polygon-contracts/blob/da2e7ee19ab8552ed278f9e3a3a25be57b8068cc/contracts/NodeOperatorRegistry.sol#L9)` – global

import of path ./interfaces/INodeOperatorRegistry.sol is not allowed. Specify names to import individually or bind all exports of the module into a name (import "path" as Name)

- [NodeOperatorRegistry.sol:10](https://github.com/lidofinance/polygon-contracts/blob/da2e7ee19ab8552ed278f9e3a3a25be57b8068cc/contracts/NodeOperatorRegistry.sol#L10) – global import of path ./interfaces/IStMATIC.sol is not allowed. Specify names to import individually or bind all exports of the module into a name (import "path" as Name)

Error/function-max-lines

- NodeOperatorRegistry.sol:90 – Function body contains 41 lines but allowed no more than 40 lines
- NodeOperatorRegistry.sol:410 – Function body contains 74 lines but allowed no more than 40 lines
- NodeOperatorRegistry.sol:496 – Function body contains 57 lines but allowed no more than 40 lines
- NodeOperatorRegistry.sol:566 – Function body contains 60 lines but allowed no more than 40 lines
- NodeOperatorRegistry.sol:635 – Function body contains 42 lines but allowed no more than 40 lines
- NodeOperatorRegistry.sol:690 – Function body contains 111 lines but allowed no more than 40 lines

Error/code-complexity

- NodeOperatorRegistry.sol:410 – Function has cyclomatic complexity 6 but allowed no more than 5
- NodeOperatorRegistry.sol:690 – Function has cyclomatic complexity 6 but allowed no more than 5
- NodeOperatorRegistry.sol:956 – Function has cyclomatic complexity 6 but allowed no more than 5

7. Appendix B. Slither



High/High/uninitialized-state

StMATIC.token2WithdrawRequest is never initialized. It is used in: – StMATIC.claimTokens(uint256) – StMATIC._claimTokensV1(uint256) – StMATIC.getMaticFromTokenId(uint256)

Informational/High/unused-state

StMATIC.recovered is never used in StMATIC

Informational/Medium/similar-names

Variable NodeOperatorRegistry.getNodeOperatorStatus(uint256)._validatorId is too similar to NodeOperatorRegistry.validatorIds

Variable NodeOperatorRegistry._getOperatorStatusAndValidator(uint256)._validatorId is too similar to NodeOperatorRegistry.validatorIds

Variable NodeOperatorRegistry.setMaxWithdrawPercentagePerRebalance(uint256)._newMaxWithdrawPercentagePerRebalance is too similar to NodeOperatorRegistry.setMaxWithdrawPercentagePerRebalance(uint256)._oldMaxWithdrawPercentagePerRebalance

Variable NodeOperatorRegistry.addNodeOperator(uint256,address)._validatorId is too similar to NodeOperatorRegistry.validatorIds

Variable NodeOperatorRegistry.getNodeOperator(uint256)._validatorId is too similar to NodeOperatorRegistry.validatorIds

Variable NodeOperatorRegistry.removeInvalidNodeOperator(uint256)._validatorId is too similar to NodeOperatorRegistry.validatorIds

Variable NodeOperatorRegistry.removeNodeOperator(uint256)._validatorId is too similar to NodeOperatorRegistry.validatorIds

Variable NodeOperatorRegistry._removeOperator(uint256,address,address)._validatorId is too similar to NodeOperatorRegistry.validatorIds

Low/Medium/calls-loop

StMATIC._calculateValidatorShares(address,uint256) has external calls inside a loop: exchangeRatePrecision = _getExchangeRatePrecision(validatorShare.validatorId())

StMATIC._calculateValidatorShares(address,uint256) has external calls inside a loop: rate = validatorShare.exchangeRate()

<p><u>StMATIC.getTotalStake(IValidatorShare)</u> has external calls inside a loop: <u>_validatorShare.getTotalStake(address(this))</u>.</p>
<p><u>StMATIC._createWithdrawRequest(address,uint256)</u> has external calls inside a loop: <u>stMaticWithdrawRequest.push(RequestWithdraw(0,IValidatorShare(_validatorShare).unbondNonces(address(this)),stakeManager.epoch() + stakeManager.withdrawalDelay(),_validatorShare))</u></p>
<p><u>StMATIC._getMaticFromRequestData(IStMATIC.RequestWithdraw)</u> has external calls inside a loop: <u>unbond = validatorShare.unbonds_new(address(this),requestData.validatorNonce)</u></p>
<p><u>StMATIC._getMaticFromRequestData(IStMATIC.RequestWithdraw)</u> has external calls inside a loop: <u>withdrawExchangeRate = validatorShare.withdrawExchangeRate()</u></p>
<p><u>StMATIC.distributeRewards()</u> has external calls inside a loop: <u>stMaticReward = validatorShare.getLiquidRewards(address(this))</u></p>
<p><u>StMATIC.distributeRewards()</u> has external calls inside a loop: <u>rewardThreshold = validatorShare.minAmount()</u></p>
<p><u>StMATIC.distributeRewards()</u> has external calls inside a loop: <u>validatorShare.withdrawRewards()</u></p>
<p><u>StMATIC.buyVoucher(address,uint256,uint256)</u> has external calls inside a loop: <u>amountSpent = IValidatorShare(_validatorShare).buyVoucher(_amount,_minSharesToMint)</u></p>
<p><u>StMATIC._getMaticFromRequestData(IStMATIC.RequestWithdraw)</u> has external calls inside a loop: <u>exchangeRatePrecision = _getExchangeRatePrecision(validatorShare.validatorId())</u></p>
<p><u>StMATIC.sellVoucher_new(address,uint256,uint256)</u> has external calls inside a loop: <u>IValidatorShare(_validatorShare).sellVoucher_new(_claimAmount,_maximumSharesToBurn)</u></p>
<p><u>NodeOperatorRegistry.getProtocolStats()</u> has external calls inside a loop: <u>(amount) = IValidatorShare(validator.contractAddress).getTotalStake(address(stMATIC))</u></p>
<p><u>NodeOperatorRegistry.listDelegatedNodeOperators()</u> has external calls inside a loop: <u>! IValidatorShare(validator.contractAddress).delegation()</u></p>
<p><u>NodeOperatorRegistry._getOperatorStatusAndValidator(uint256)</u> has external calls inside a loop: <u>validator = stakeManager.validators(_validatorId)</u></p>

Low/Medium/missing-zero-check

<p><u>StMATIC.initialize(address,address,address,address,address,address,address)</u>. <u>_dao</u> lacks a zero-check on : - <u>dao = _dao</u></p>
<p><u>StMATIC.initialize(address,address,address,address,address,address,address)</u>. <u>_insurance</u> lacks a zero-check on : - <u>insurance = _insurance</u></p>
<p><u>StMATIC.setDaoAddress(address)</u>. <u>_newDAO</u> lacks a zero-check on : - <u>dao = _newDAO</u></p>
<p><u>StMATIC.setInsuranceAddress(address)</u>. <u>_address</u> lacks a zero-check on : - <u>insurance = _address</u></p>
<p><u>StMATIC.initialize(address,address,address,address,address,address,address)</u>. <u>_token</u> lacks a zero-check on : - <u>token = _token</u></p>

Low/Medium/reentrancy–benign

<p>Reentrancy in <u>StMATIC._createWithdrawRequest(address,uint256)</u>: External calls: – <u>sellVoucher_new(_validatorShare,amount,type()(uint256).max)</u> – <u>IValidatorShare(_validatorShare).sellVoucher_new(_claimAmount,_maximumSharesToBurn)</u> State variables written after the call(s): – <u>stMaticWithdrawRequest.push(RequestWithdraw(0,IValidatorShare(_validatorShare).unbondNonces(address(this)),stakeManager.epoch() + stakeManager.withdrawalDelay(),_validatorShare))</u></p>
<p>Reentrancy in <u>StMATIC.claimTokensFromValidatorToContract(uint256)</u>: External calls: – <u>unstakeClaimTokens_new(lidoRequest.validatorAddress,lidoRequest.validatorNonce)</u> – <u>IValidatorShare(_validatorShare).unstakeClaimTokens_new(_unbondNonce)</u> State variables written after the call(s): – <u>totalBuffered += claimedAmount</u></p>
<p>Reentrancy in <u>StMATIC._claimTokensV1(uint256)</u>: External calls: – <u>poLidoNFT.burn(_tokenId)</u> State variables written after the call(s): – <u>reservedFunds -= amountToClaim</u> – <u>totalBuffered -= amountToClaim</u></p>
<p>Reentrancy in <u>StMATIC._requestWithdraw(uint256,address,uint256,uint256)</u>: External calls: – <u>sellVoucher_new(validatorShare,amount2WithdrawFromValidator,type()(uint256).max)</u> – <u>IValidatorShare(_validatorShare).sellVoucher_new(_claimAmount,_maximumSharesToBurn)</u> State variables written after the call(s): – <u>token2WithdrawRequests[tokenId].push(RequestWithdraw(0,IValidatorShare(validatorShare).unbondNonces(address(this))),stakeManager.epoch() + stakeManager.withdrawalDelay(),validatorShare))</u></p>
<p>Reentrancy in <u>NodeOperatorRegistry._removeOperator(uint256,address,address)</u>: External calls: – <u>stMATIC.withdrawTotalDelegated(_contractAddress)</u> State variables written after the call(s): – <u>delete validatorIdToRewardAddress[_validatorId]</u> – <u>delete validatorRewardAddressTold[_rewardAddress]</u></p>

Low/Medium/reentrancy–events

<p>Reentrancy in <u>StMATIC._claimTokensV2(uint256)</u>: External calls: – <u>poLidoNFT.burn(_tokenId)</u> – <u>unstakeClaimTokens_new(usersRequest[idx].validatorAddress,usersRequest[idx].validatorNonce)</u> – <u>IValidatorShare(_validatorShare).unstakeClaimTokens_new(_unbondNonce)</u> – <u>IERC20Upgradeable(token).safeTransfer(msg.sender,amountToClaim)</u> Event emitted after the call(s): – <u>ClaimTokensEvent(msg.sender,_tokenId,amountToClaim,0)</u></p>
<p>Reentrancy in <u>StMATIC._claimTokensV1(uint256)</u>: External calls: – <u>poLidoNFT.burn(_tokenId)</u> – <u>unstakeClaimTokens_new(usersRequest.validatorAddress,usersRequest.validatorNonce)</u> – <u>IValidatorShare(_validatorShare).unstakeClaimTokens_new(_unbondNonce)</u> – <u>IERC20Upgradeable(token).safeTransfer(msg.sender,amountToClaim)</u> Event emitted after the call(s): – <u>ClaimTokensEvent(msg.sender,_tokenId,amountToClaim,0)</u></p>

Medium/High/incorrect–equality

<p><u>StMATIC._convertMaticToStMatic(uint256,uint256)</u> uses a dangerous strict equality: – <u>totalStMaticSupply == 0</u></p>
<p><u>StMATIC._convertStMaticToMatic(uint256,uint256)</u> uses a dangerous strict equality: – <u>totalStMaticSupply == 0</u></p>
<p><u>StMATIC._convertMaticToStMatic(uint256,uint256)</u> uses a dangerous strict equality: – <u>totalPooledMatic == 0</u></p>
<p><u>StMATIC.rebalanceDelegatedTokens()</u> uses a dangerous strict equality: – <u>shares == 0</u></p>

StMATIC._requestWithdrawUnbalanced(uint256,INodeOperatorRegistry.ValidatorData[],uint256,uint256[],uint256[],uint256) uses a dangerous strict equality: – currentAmount2WithdrawInMatic == 0

StMATIC._convertStMaticToMatic(uint256,uint256) uses a dangerous strict equality: – _totalPooledMatic == 0

StMATIC.withdrawTotalDelegated(address) uses a dangerous strict equality: – shares == 0

StMATIC.delegate() uses a dangerous strict equality: – shares == 0

Medium/Medium/divide-before-multiply

StMATIC.distributeRewards() performs a multiplication on the result of a division: – protocolRewards = totalRewards * protocolFee / 100 – operatorsRewards = (protocolRewards * entityFees.operators) / 100

StMATIC.distributeRewards() performs a multiplication on the result of a division: – protocolRewards = totalRewards * protocolFee / 100 – insuranceRewards = (protocolRewards * entityFees.insurance) / 100

StMATIC.distributeRewards() performs a multiplication on the result of a division: – protocolRewards = totalRewards * protocolFee / 100 – daoRewards = (protocolRewards * entityFees.dao) / 100

NodeOperatorRegistry.getValidatorsDelegationAmount(uint256) performs a multiplication on the result of a division: – rebalanceTarget = (totalStaked + _amountToDelegate) / totalActiveNodeOperator – (rebalanceTarget * 100) / stakePerOperator[idx] >= distanceThresholdPercents

Medium/Medium/reentrancy-no-eth

Reentrancy in StMATIC.claimTokensFromValidatorToContract(uint256): External calls: – unstakeClaimTokens_new(lidoRequest.validatorAddress,lidoRequest.validatorNonce) – IValidatorShare(_validatorShare).unstakeClaimTokens_new(_unbondNonce) State variables written after the call(s): – stMaticWithdrawRequest[_index] = stMaticWithdrawRequest[length – 1] StMATIC.stMaticWithdrawRequest can be used in cross function reentrancies: – StMATIC._createWithdrawRequest(address,uint256) – StMATIC.calculatePendingBufferedTokens() – StMATIC.getTotalWithdrawRequest() – StMATIC.stMaticWithdrawRequest – stMaticWithdrawRequest.pop() StMATIC.stMaticWithdrawRequest can be used in cross function reentrancies: – StMATIC._createWithdrawRequest(address,uint256) – StMATIC.calculatePendingBufferedTokens() – StMATIC.getTotalWithdrawRequest() – StMATIC.stMaticWithdrawRequest

Reentrancy in StMATIC.delegate(): External calls: – IERC20Upgradeable(token).safeApprove(address(stakeManager),0) – IERC20Upgradeable(token).safeApprove(address(stakeManager),amountToDelegate) – buyVoucher(_validatorAddress,amountToDelegatePerOperator,0) – amountSpent = IValidatorShare(_validatorShare).buyVoucher(_amount,_minSharesToMint) State variables written after the call(s): – totalBuffered = remainder + lreservedFunds StMATIC.totalBuffered can be used in cross function reentrancies: – StMATIC._claimTokensV1(uint256) – StMATIC._claimTokensV2(uint256) – StMATIC._getTotalPooledMatic(uint256) – StMATIC.rebalanceDelegatedTokens() – StMATIC.totalBuffered

Reentrancy in StMATIC.distributeRewards(): External calls: – validatorShare.withdrawRewards() – IERC20Upgradeable(token).safeTransfer(dao,daoRewards) – IERC20Upgradeable(token).safeTransfer(insurance,insuranceRewards) – IERC20Upgradeable(token).safeTransfer(operatorInfos[i_scope_0].rewardAddress,operatorReward) State variables written after the call(s): – totalBuffered = currentBalance StMATIC.totalBuffered can be used in cross function

reentrancies: - StMATIC. claimTokensV1(uint256) - StMATIC. claimTokensV2(uint256) - StMATIC. getTotalPooledMatic(uint256) - StMATIC.rebalanceDelegatedTokens() - StMATIC.totalBuffered

Reentrancy in StMATIC.requestWithdraw(uint256,address): External calls: - tokenId = poLidoNFT.mint(msg.sender) - currentAmount2WithdrawInMatic =
requestWithdrawUnbalanced(tokenId,activeNodeOperators,bigNodeOperatorLength,bigNodeOperatorIds,allowedAmountToRequestFromOperators,currentAmount2WithdrawInMatic) - IValidatorShare(validatorShare).sellVoucher_new(_ claimAmount,_ maximumSharesToBurn) - currentAmount2WithdrawInMatic =
requestWithdrawUnbalanced(tokenId,activeNodeOperators,smallNodeOperatorLength,smallNodeOperatorIds,allowedAmountToRequestFromOperators,currentAmount2WithdrawInMatic) - IValidatorShare(validatorShare).sellVoucher_new(_ claimAmount,_ maximumSharesToBurn) State variables written after the call(s): - currentAmount2WithdrawInMatic =
_requestWithdrawUnbalanced(tokenId,activeNodeOperators,smallNodeOperatorLength,smallNodeOperatorIds,allowedAmountToRequestFromOperators,currentAmount2WithdrawInMatic) - token2WithdrawRequests[tokenId].push(RequestWithdraw(0,IValidatorShare(validatorShare).unbondNonces(address(this)),stakeManager.epoch() + stakeManager.withdrawalDelay(),validatorShare)) StMATIC.token2WithdrawRequests can be used in cross function reentrancies: - StMATIC. claimTokensV2(uint256) - StMATIC.claimTokens(uint256) - StMATIC.getMaticFromTokenId(uint256) - StMATIC.getToken2WithdrawRequests(uint256) - StMATIC.token2WithdrawRequests

Reentrancy in StMATIC. claimTokensV2(uint256): External calls: - poLidoNFT.burn(_ tokenId) - unstakeClaimTokens_new(usersRequest[idx].validatorAddress,usersRequest[idx].validatorNonce) - IValidatorShare(_ validatorShare).unstakeClaimTokens_new(_ unbondNonce) State variables written after the call(s): - reservedFunds -= _ amountToClaim StMATIC.reservedFunds can be used in cross function reentrancies: - StMATIC. claimTokensV1(uint256) - StMATIC. claimTokensV2(uint256) - StMATIC. getTotalPooledMatic(uint256) - StMATIC.rebalanceDelegatedTokens() - StMATIC.reservedFunds - totalBuffered -= _ amountToClaim StMATIC.totalBuffered can be used in cross function reentrancies: - StMATIC. claimTokensV1(uint256) - StMATIC. claimTokensV2(uint256) - StMATIC. getTotalPooledMatic(uint256) - StMATIC.rebalanceDelegatedTokens() - StMATIC.totalBuffered

Medium/Medium/uninitialized-local

- StMATIC. claimTokensV2(uint256).amountToClaim is a local variable never initialized
- NodeOperatorRegistry._ getValidatorsDelegationInfos().maxAmount is a local variable never initialized
- NodeOperatorRegistry._ getValidatorsDelegationInfos().minAmount is a local variable never initialized

Optimization/High/constable-states

- StMATIC.recovered should be constant
- StMATIC.submitThreshold should be constant
- StMATIC.lastWithdrawnValidatorId should be constant
- StMATIC.submitHandler should be constant

8. Appendix C. Tests



Tests result

169 passing (2m)

Tests coverage

- Solc version: 0.8.7
- Optimizer enabled: true
- Runs: 200
- Block limit: 30000000 gas

Contract	Method	Min	Max	Avg	# calls
ERC20Upgradeable	approve	31692	106190	59552	76
ERC20Upgradeable	transferFrom	93893	130984	106707	6
NodeOperatorRegistry	addNodeOperator	145641	162753	152418	288
NodeOperatorRegistry	exitNodeOperatorRegistry	-	-	75870	3
NodeOperatorRegistry	grantRole	58464	58860	58761	4
NodeOperatorRegistry	pause	54322	54346	54328	4
NodeOperatorRegistry	removeInvalidNodeOperator	78480	86658	82583	8
NodeOperatorRegistry	removeNodeOperator	77819	296362	188269	47
NodeOperatorRegistry	revokeRole	36433	36829	36730	4
NodeOperatorRegistry	setDistanceThreshold	34878	37690	37383	114
NodeOperatorRegistry	setMaxWithdrawPercentagePerRebalance	34821	37621	37310	9
NodeOperatorRegistry	setMinRequestWithdrawRange	-	-	37682	14
NodeOperatorRegistry	setRewardAddress	-	-	60847	2
NodeOperatorRegistry	setStMaticAddress	37945	37957	37956	80
NodeOperatorRegistry	setVersion	43256	98946	61819	3
NodeOperatorRegistry	unpause	-	-	32422	1

Contract	Method	Min	Max	Avg	# calls
PoLidoNFT	burn	50937	85284	60910	6
PoLidoNFT	mint	118106	149506	130392	23
PoLidoNFT	setStMATIC	36370	53482	53265	79
PoLidoNFT	togglePause	-	-	54228	1
Polygon	approve	46196	46268	46260	155
Polygon	mint	33659	67931	56565	298
Polygon	transfer	29714	51614	50568	118
SelfDestructor	selfdestruct	-	-	29289	1
StakeManagerMock	setEpoch	26440	43540	41310	23
StakeManagerMock	slash	-	-	31735	17
StakeManagerMock	stakeFor	1025920	1043032	1032839	292
StakeManagerMock	unjail	-	-	26873	1
StakeManagerMock	unstake	-	-	43711	16
StakeManagerMock	unstakeClaim	-	-	42409	5
StakeManagerMock	updateCommissionRate	-	-	43823	2
StMATIC	claimTokens	129549	290144	167278	36
StMATIC	claimTokensFromValidatorToContract	115503	198234	147988	23
StMATIC	delegate	200119	1188178	399633	57
StMATIC	distributeRewards	217296	607087	380485	10
StMATIC	rebalanceDelegatedTokens	618950	783893	701422	2
StMATIC	recover	-	-	144041	1
StMATIC	requestWithdraw	282588	1546441	546862	124
StMATIC	setDaoAddress	-	-	37966	1
StMATIC	setDelegationLowerBound	34657	54785	47692	14

Contract	Method	Min	Max	Avg	# calls
StMATIC	setFxStateRootTunnel	55031	55043	55042	78
StMATIC	setInsuranceAddress	-	-	37796	2
StMATIC	setNodeOperatorRegistryAddress	-	-	37722	2
StMATIC	setProtocolFee	37853	54953	53923	83
StMATIC	setRewardDistributionLowerBound	-	-	54824	2
StMATIC	submit	115120	478810	235254	155
StMATICMock	setOperator	43844	43856	43855	75
ValidatorShareMock	increaseStakeFor	23670	43666	42684	103
ValidatorShareMock	setExchangeRate	26599	26611	26606	5
ValidatorShareMock	slash	47814	67714	54633	15
ValidatorShareMock	updateDelegation	-	-	26618	9
Deployments				% of limit	
ERC721Test	-	-	1149674	3.8 %	
FxBaseRootMock	-	-	119047	0.4 %	
NodeOperatorRegistry	-	-	3703135	12.3 %	
PoLidoNFT	-	-	2211492	7.4 %	
Polygon	-	-	653392	2.2 %	
SelfDestructor	-	-	93135	0.3 %	
StakeManagerMock	1749030	1749042	1749041	5.8 %	
StMATIC	-	-	5349662	17.8 %	
StMATICMock	-	-	151885	0.5 %	

STATE MIND