LIDO

Lido for Polygon

Smart Contracts

Security Audit Report

March 10, 2022

0XORIO

LIST OF CONTENTS

1 Executive Summary

This report consists of the audit results performed by Oxorio team on the Lido for Polygon project, at the request of the Lido team. The audited code can be found in the public Lido for Polygon Github Repository. Moreover, we used multiple versions to compose this report according to relevant commits. Versioning methodology is explained in the Methodology part.

The main goals of this audit are:

- to review Lido for Polygon's solidity implementation for its decentralized staking model,
- to study potential security vulnerabilities, its general design and architecture,
- to uncover errors and bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations which could improve its quality as a whole.

1.1 Disclaimer

Note that as of the date of publishing, the contents of this document reflect the current understanding of investigated security patterns and the state of art regarding smart contract security. Given the size of the project, the findings detailed here are not to be considered exhaustive. Further testing and auditing are recommended after the covered issues would be fixed.

1.2 Methodology

On the methodology part, we do the following audit steps:

1. Manual code study
Manually code study 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. Audit report publication on the official website
The final audit version is provided to the client and also published on the official website of the company.

1.3 Structure of the Document

This report contains the list of issues and comments divided by version and also by their severity and status levels. Each issue is aligned with the code file that it is represented in for the readability of the report. For an easy way of navigation, a table of contents is provided at the beginning of the report.

1.4 Documentation

For this audit, the following sources of truth about how the Lido for Polygon smart contracts should work were used:

- main GitHub repository of the project
- Almanac documentation provided by the client.

These were considered the specification, and when discrepancies arose with the actual code behaviour, there were consultations directly with the Lido team.

1.5 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.

Clients include Lido, among others.

More info at: oxor.io

2 Audit Scope

Audit includes 6 revisions which are presented in the following Table:

Table: Six Revisions

Revision №	Hash Commit	Date
Revision 1	05bff82670a76eb2ccf1c66d5cb242bbb375d40d [PR#33]	10/12/2021
Revision 2	431300f0ebcad04376c2a90609519eb2ba410b70	11/01/2022
Revision 3	0f50eebbc3d36a77d0b166963427c576fd5a3765 [PR#36]	18/01/2022
Revision 4	bda8645bff4916a12d21fff38df8379e036724fd [PR#42]	28/01/2022
Revision 5	511da0d53549e29145689f09a6666d3bd3c6c224 [PR#49]	04/02/2022
Revision 6	6e2ac6a5b93cecfcbd77f2a0c4e5021e30533741 [PR#56]	16/02/2022

Besides this, the next commits including pull requests with minor fixes were checked as well. No issues were found.

- 0b1ce58cce24cbc6d5b5ad48cde2c765ba2e544a [PR#57]
- F4e7f53ac6b3a38197f6153b3b3e1b23f4f074cd [PR#58]
- Ea33825882c188943d3f85b552e33322842406e2 [PR#59]
- C383ab54802816d7a0ba516728d269c9028a35e4 [PR#60]
- 2f4d94b116ba4a160c1c007a8e3870c3ad8aba1e [PR#61]
- 5be86d9700201397b2fcff13daaaf9549cfe0272 [PR#62]

3 Severity Level Reference

Every issue in this report was assigned a severity level from the following:

CRITICAL

Critical severity issues need to be fixed as soon as possible. Can result in loss of funds by the usage of the project's contract(s).

MAJOR

Bugs that can contribute to contracting failure or other major operational issues.

WARNING

High severity issues will probably bring problems and should be fixed. Can expose contracts to various attacks.

INFO

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

4 Status Level Reference

Every issue in this report was assigned a status level from the following:

FIXED

This indicates that the issue was fixed by the client.

ACKNOWLEDGED

This indicates that the issue was acknowledged by the client.

DISMISSED

This indicates that the issue was dismissed by the client.

5 Revision 1: Findings

5.1 CRITICAL

5.1.1 Withdrawal of Less Matic Than Their stMatic Value

SEVERITY	CRITICAL
STATUS	FIXED

Description

Steps to reproduce the error:

1. User calls requestWithdraw
   `(totalDelegated > currentAmount2WithdrawInMatic) &&`
   `(amount2WithdrawFromValidator < totalAmount2WithdrawInMatic)`
   If these conditions are true we are in the while loop #L178 at least twice;

2. From the second and larger iterations, `token2WithdrawRequest[tokenId]` would be overwritten #L224;

3. User waits and then calls `claimTokens;`

4. A user gets `amount2WithdrawFromValidator` only from the last iteration less than stMatic value.

Recommendation

To rewrite the logic so `token2WithdrawRequest[tokenId]` is not rewritten.

5.1.2 Delegation and Distribute Rewards with Decrease of totalBuffered

SEVERITY	CRITICAL
STATUS	FIXED

Description

Steps to reproduce the error:

1. User calls submit function, `totalBuffered` increases by `_amount`;

2. User calls the delegate function, and in the worst case, totalBuffered is not changed. In addition to that, the same error occurs in cases where only some tokens are not delegated. For example:
   - a user submits 1,000,000 MATIC
   - there are 10 validators
   - maxDelegateLimit is default, 10 MATIC
   - `reservedFunds` = 0
   a. `availableAmountToDelegate` = 1000000 MATIC;
   b. `maxDelegateLimitsSum` = 100;
   c. `totalToDelegatedAmount` = 100;
   d. `remainder` = 1000000 - 100 = 999900;
   e. `totalBuffered` = 999900;
   f. Matic balance of this contract is 999,900 MATIC;

3. User calls requestWithdraw function
   a. `(totalDelegated > currentAmount2WithdrawInMatic)` we go in the while loop #L178.
      NOTE: there is a critical bug with `token2WithdrawRequest[tokenId]` in this while loop, it's ignored in this example;
   b. `token2WithdrawRequest[tokenId]` is saved with `validatorAddress != address(0)` #L224;

4. User waits and calls claimTokens function
   a. We are inside if statement on #L347, totalBuffered is not changed, but MATIC sent to the user #L371. So Matic balance of this contract is 0, but totalBuffered = 999900;
   b. totalBuffered is invalid now, it's more than the MATIC balance of the contract. And it can be overwritten only in the delegate function;

5. delegate function is called:
   a. require (totalBuffered > delegationLowerBound) false #L256 availableAmountToDelegate incorrect, > MATIC balance of this contract (availableAmountToDelegate=990000, balance is 0) #L268 totalToDelegatedAmount = availableAmountToDelegate most of the times (totalToDelegatedAmount=990000) #L276;
   b. stakeManager gets the approve;
   c. amountToDelegatePerOperator is incorrect. It is greater than needed;
      a. maxDelegateLimit by default is 10 MATIC, recall that in our example there are 10 validators amountToDelegatePerOperator = 10 * 990k / 10 * 10 = 99k;
      b. In our example any amountToDelegatePerOperator > 0 will lead to revert;
      c. In a general case any (totalBuffered - reservedFunds) > balance of MATIC will lead to a revert;
   d. buyVoucher transfers too many MATIC and reverts when this contract's balance of MATIC is insufficient (buyVoucher -> ValidatorShare.buyVoucher -> stakeManager.delegationDeposit -> token.transferFrom).

Results:

- delegate is locked

- distributeRewards is locked - overflow at #L389.

Recommendation

To rewrite the logic so totalBuffered is always in sync with the real buffered value.

5.1.3 Function Lock due to Validator's Slash

SEVERITY	CRITICAL
STATUS	FIXED

Description

Validator's slash between requestWithdraw and claimTokens calls leads to the function lock.
Steps to reproduce the error:

1. User calls requestWithdraw(Y)
   a. We calculate allowedAmount2Withdraw = X based on Y value and save it in a token2WithdrawRequest[tokenId].amountToClaim;

2. Validator was slashed contracts/staking/validatorShare/ValidatorShare.sol #L195
   a. In slash method withdrawPool value is decreased;

3. User calls claimTokens
   a. unstakeClaimTokens_new is called #L348;
   b. The calculated amount in ValidatorShare._unstakeClaimTokens (L304) would be lower than the original amount X, saved at step 1.a because withdrawPool was decreased due to slashing;
   c. LidoMatic contract receives less than X tokens.

4. LidoMatic tries to transfer X tokens, which may fail, the contract may not have enough balance #L371;

5. If it does not fail the user receives funds that s/he should not have;

6. totalDelegated value is out of sync #L353 with the contract balance now, it might be more than the contract's balance
   a. delegate function will fail, incorrect calculations of availableAmountToDelegate at #L268 will lead to reverts on buyVoucher call or tokens that is not supposed to be;
   b. totalPooledMatic is incorrect #L138 => amountToMint incorrect => submitter receives less than he should;
   c. distributeRewards will fail or too many rewards will be distributed.

Recommendation

To rewrite the logic so slashing is handled appropriately.

5.1.4 Incorrect Usage of approvalExists Mapping

SEVERITY	CRITICAL
STATUS	FIXED

Description

#L58

approvalExists mapping should use tokenId as a keys, but now it uses array indexes which leads to incorrect contract behavior and transaction revertions during approvals.

Recommendation

To use tokenId as key for approvalExists mapping.

5.2 WARNING

5.2.1 Division by Zero if maxDelegateLimit is Zero

SEVERITY	WARNING
STATUS	FIXED

Description

If all operators have zero maxDelegateLimit then maxDelegateLimitsSum would be zero (LidoMatic.sol#L273). This leads to division by zero at LidoMatic.sol#L290.

Recommendation

Make sure maxDelegateLimit is never equal zero for all operators or make a check before division.

5.2.2 Possible Overflow if maxDelegateLimit is Too Large

SEVERITY	WARNING
STATUS	FIXED

Description

#L288

If operatorShares[i].maxDelegateLimit set too high an overflow is possible. For example, errors can happen when there is no limit.

Recommendation

To make sure maxDelegateLimit is never too high. A check in nodeOperator.setMaxDelegateLimit. could be added.

5.2.3 Possible Denial of Service if the Number of Operators is Too High

SEVERITY	WARNING
STATUS	ACKNOWLEDGED

Description

LidoMatic uses a lot of loops through operator shares. If the number of operators would be significant it can potentially lead to out of gas errors.

Recommendation

To decrease the number of loops and optimize gas consumption or limit the possible number of operators.

5.2.4 Potential Reentrancy Attack

SEVERITY	WARNING
STATUS	FIXED

Description

#L371

State is changed after an external call. It can lead to potential reentrancy attacks.

Recommendation

To use Checks Effects Interactions pattern to prevent possible reentrancy attacks. It's better to place external calls after state modification at L373 to prevent reentrancy in the future. (Verification is in line L337)

5.2.5 Empty Slots in operatorShares Array.

SEVERITY	WARNING
STATUS	FIXED

Description

#L942

The resulting array operatorShares will contain empty slots if some operators have non STAKED status. It may lead to bugs in further usage of this function in external contracts. For example LidoMatic doesn't checks operatorShares for empty values when calling nodeOperator.getOperatorShares().

Recommendation

To use an extra counter for filling operatorShares array without empty slots.

5.2.6 index Variable is Unused

SEVERITY	WARNING
STATUS	FIXED

Description

#L991

index is only incremented but not used. Should be used at line L985.

Recommendation

To use the index counter while filling an array of rewardAddresses.

5.2.7 Auth Modifier is a Copy of AccessControlUpgradeable.onlyRole

SEVERITY	WARNING
STATUS	FIXED

Description

Functional auth modifier copies onlyRole inherited from OpenZeppelin's AccessControlUpgradeable contact.

Recommendation

To remove the auth modifier and use onlyRole from OpenZeppelin.

5.2.8 Subtraction with Overflow

SEVERITY	WARNING
STATUS	FIXED

Description

#L372

The only place where totalTimesValidatorsSlashed is updated is subtraction. This will lead to overflow and empty revert messages.

Recommendation

To redesign totalTimesValidatorsSlashed usage.

5.2.9 Usage of _mint instead of _safeMint

SEVERITY	WARNING
STATUS	DISMISSED

Description

#L44

Usage of _mint instead of _safeMint.

According to OpenZeppelin's documentation usage of _mint method is discouraged, _safeMint should be used whenever possible.

Recommendation

To use _safeMint instead of _mint.

5.2.10 Contract Without Constructor

SEVERITY	WARNING
STATUS	DISMISSED

Description

#L7

Contract has no constructor. There is no way the state variable would be initialized. It is used in isOperator modifier and is required for all methods that update state. No contract that inherits Validator.sol found and it's unclear how it will work.

Recommendation

To initialize state variable properly.

5.3 INFO

5.3.1 Multiple Initialization Problem

SEVERITY	INFO
STATUS	FIXED

Description

#L94

__ERC20_init("Staked MATIC", "StMATIC");

Recommendation

To use ***_`init_unchained` functions for initialization as suggested by OpenZeppelin.

5.3.2 Missing notPaused Modifier

SEVERITY	INFO
STATUS	FIXED

Description

Missing notPaused modifier (isStoped in docs) in the majority of functions, except requestWithdraw.

Recommendation

To add `notPaused` modifier if needed.

5.3.3 Incorrect Functions Visibility

SEVERITY	INFO
STATUS	FIXED

Description

Examples: #L650 and #L591

Recommendation

Functions that are not used inside the contract should be declared as external to explicitly show that it's not used inside the contract. Moreover, this will optimize gas consumption in some cases.

5.3.4 Commented Code

SEVERITY	INFO
STATUS	FIXED

Description

Example in #L42.

Recommendation

The best practice is to remove all commented code.

5.3.5 No Tight Variable Packing in FeeDistribution Struct

SEVERITY	INFO
STATUS	FIXED

Description

#L65

Tight packing can be applied to a struct FeeDistribution - now it uses uint256 for its members, but the maximum value for them is only 100 because it's a percentage value. See tight variable packing pattern.

Recommendation

To use uint8 instead of uint256 for FeeDistribution members.

5.3.6 No Zero Address Checks in the Initializer

SEVERITY	INFO
STATUS	DISMISSED

Description

#L97

Potential brick of the contract in the initializer.

Recommendation

To add a check that addresses are not 0 to not accidentally brick the contract in the initializer, at least for DAO.

5.3.7 Contradiction with the Docs in Access Roles Distribution

SEVERITY	INFO
STATUS	FIXED

Description

In the documentation there is the following roles description
MANAGE_FEE: fee manager role (initially set to msg.sender)
...
SET_TREASURY: treasurer role (initially set to msg.sender)
GOVERNANCE: DAO role (initially set to msg.sender)
But in fact MANAGE_FEE and SET_TREASURY are set to _dao, GOVERNANCE is changed to DAO and set to _dao #L99

Recommendation

To update the docs or change the code accordingly.

5.3.8 Contradiction with the Docs in claimTokens Function

SEVERITY	INFO
STATUS	FIXED

Description

#L338

According to the documentation it should be:
block.timestamp > userRequests.requestTime + stakeManager.withdrawalDelay()

Recommendation

To update the docs or change the code accordingly.

5.3.9 Equality Instead of Subtraction

SEVERITY	INFO
STATUS	FIXED

Description

May use just == instead of subtraction and comparing with zero.

Recommendation

To replace to if(currentAmount2WithdrawInMatic == amount2WithdrawFromValidator) for code readability.

5.3.10 Unused Variable validator2Nonce

SEVERITY	INFO
STATUS	FIXED

Description

Unused variable in #L47:
mapping(address => uint256) public validator2Nonce; // DELETE before deploying to production

Recommendation

To remove the unused variable.

5.3.11 Writing to the Storage in a Loop

SEVERITY	INFO
STATUS	FIXED

Description

#L298 writing to storage in the loop.

Recommendation

To use a local variable in the loop to calculate the total amountToDelegatePerOperator value and use it after to update the storage only once. Writing to storage in most cases is a very expensive operation.

5.3.12 Missing Version Function

SEVERITY	INFO
STATUS	FIXED

Description

LidoMatic doesn't have a version function while contracts NodeOperatorRegistry.sol, Validator.sol and ValidatorFactory.sol have.

Recommendation

To add version function to LidoMatic contract.

5.3.13 Usage of array.length in Loops

SEVERITY	INFO
STATUS	FIXED

Description

Reading the length of a storage array costs more than reading a memory variable, e.g.:

NodeOperatorRegistry.sol#L360

NodeOperatorRegistry.sol#L691

NodeOperatorRegistry.sol#L939

Recommendation

To use uint256 length = array.length; in loop for (uint i = 0; i < length; i++){}

5.3.14 No Events Usage in LidoMatic Contract

SEVERITY	INFO
STATUS	FIXED

Description

LidoMatic doesn't emit any events.

Recommendation

To consider adding events for at least such important actions as pausing/unpausing contract or user interactions for better logging and monitoring.

5.3.15 Multiple Initialization Problem

SEVERITY	INFO
STATUS	FIXED

Description

#L35

Multiple initialization problem in the line.

Recommendation

To use ***`_init_unchained` functions for initialization as suggested by OpenZeppelin.

5.3.16 Unnecessary Reads of tokenIdIndex Variable From the Storage

SEVERITY	INFO
STATUS	FIXED

Description

#L43

Unnecessary reads of tokenIdIndex variable from storage in the line.

Recommendation

To use local variable to prevent expensive reading from storage:

```
function mint(address _to) external isLido returns (uint256) {
    uint256 newTokenIdIndex = ++tokenIdIndex;
    _safeMint(_to, newTokenIdIndex);
    return newTokenIdIndex;
}
```

5.3.17 Redundant Storage Variable indexExists

SEVERITY	INFO
STATUS	FIXED

Description

#L19

Storage variable indexExists is never read and only written.

Recommendation

To remove indexExists variable if it's not used.

5.3.18 Implicit Usage of ERC721PausableUpgradeable Functional

SEVERITY	INFO
STATUS	FIXED

Description

#L81

Implicit usage of ERC721 PausableUpgradeable functional.

Recommendation

To inherit from ERC721PausableUpgradeable contract to use it's functional explicitly.

5.3.19 Duplicating Code of Resetting Approvals

SEVERITY	INFO
STATUS	FIXED

Description

#L105

Duplicating code of resetting approvals.

Recommendation

To move resetting approvals to a separate method and to use it in approve and _beforeTokenTransfer methods.

```
function _resetApprovals(uint256 tokenId, bool resetIndex) internal {
    uint256 approvedIndex = approved2Index[tokenId];
    if (approvalExists[approvedIndex]) {
        address lastApprovedAddress = getApproved(tokenId);
        uint256[] storage lastApprovedTokens = address2Approved[
            lastApprovedAddress
        ];

        delete lastApprovedTokens[approvedIndex];
        if (resetIndex) {
            approved2Index[tokenId] = 0;
            approvalExists[approvedIndex] = false;
        }
    }
}
```

5.3.20 Unnecessary Reads of approvedTokens Variable from Storage

SEVERITY	INFO
STATUS	FIXED

Description

#L94
approvedTokens is used only if approval exists, so it's better to move the declaration of approvalExists in this block.

Recommendation

To read approvalExists value from storage only if it's needed.

5.3.21 Missing Version Function

SEVERITY	INFO
STATUS	FIXED

Description

LidoNFT does not have a version function while contracts NodeOperatorRegistry.sol, Validator.sol and ValidatorFactory.sol have.

Recommendation

To add a version function.

5.3.22 Extra Calls of External "getPolygonERC20" Method

SEVERITY	INFO
STATUS	FIXED

Description

#L214

operator.getPolygonERC20() is called 3 times in a row in unstakeClaim function.

Recommendation

To save this external call result to a stack variable once and reuse it for gas saving.

5.3.23 Debug Import in Validator.sol

SEVERITY	INFO
STATUS	FIXED

Description

Validator.sol#L7

Import of hardhat/console.sol.

Recommendation

To remove debug import.

5.3.24 Mis-usage of Structure State

SEVERITY	INFO
STATUS	FIXED

Description

#L24

The reason why the structure state is used for storing the operator and validatorImplementation variables is not clear.

Recommendation

To store operator and validatorImplementation variables in the storage directly.

5.3.25 Debug Import in ValidatorFactory.sol

SEVERITY	INFO
STATUS	FIXED

Description

ValidatorFactory.sol#L5

Recommendation

To remove debug import.

5.3.26 isOwner modifier is a Copy of OwnableUpgradeable.onlyOwner

SEVERITY	INFO
STATUS	FIXED

Description

#L44

Functional of isOwner modifier copies onlyOwner inherited from OpenZeppelin's OwnableUpgradeable contract.

Recommendation

To remove the isOwner modifier and to use onlyOwner instead.

5.3.27 Unused SetValidatorImplementation Event.

SEVERITY	INFO
STATUS	FIXED

Description

#L37

Recommendation

To remove SetValidatorImplementation event or emit it.

5.3.28 Unused Storage Variable Operator.

SEVERITY	INFO
STATUS	FIXED

Description

#L12 It is unclear how operator variable should be used. It is never read and only written to.

Recommendation

To store operator and validatorImplementation variables in the storage directly.

5.3.29 No Verification for whenNotPaused modifier

SEVERITY	INFO
STATUS	FIXED

Description

Functions setOperatorName, setOperatorRewardAddress, setDefaultMaxDelegateLimit, setMaxDelegateLimit, setSlashingDelay can be called when the contract is on pause.

Recommendation

To add whenNotPaused modifier if it is required.

5.3.30 No Checks for _defaultMaxDelegateLimit Argument

SEVERITY	INFO
STATUS	FIXED

Description

#L643
_defaultMaxDelegateLimit might be equal to zero.

Recommendation

To check at least that **_defaultMaxDelegateLimit** is not equal to zero.

5.3.31 No Checks for _maxDelegateLimit argument

SEVERITY	INFO
STATUS	FIXED

Description

#L653
Needs checks that variables are not equal to zero.

Recommendation

To check at least that _maxDelegateLimit is not equal to zero. Otherwise, it can lead to division by zero in L273.

5.3.32 Redundant check _amount > 0

SEVERITY	INFO
STATUS	FIXED

Description

#L441

No need to verify that _amount > 0 because it will be automatically applied by the usage of checkStakeAmount modifier.

Recommendation

To remove excess verification require(_amount > 0, "Amount is ZERO");.

5.3.33 Nonoptimal Gas Consumption in removeOperator Function

SEVERITY	INFO
STATUS	FIXED

Description

#L343

This line could be re-written in order to reduce gas consumption.

Recommendation

It might be better to place no.status to stack variable and use it, since using sload opcode twice consumes more gas than using it once and pushing value to stack for further usage

5.3.34 Nonoptimal Gas Consumption in withdrawRewards Function

SEVERITY	INFO
STATUS	FIXED

Description

#L685

Nonoptimal gas consumption.

Recommendation

It might be better to place totalStakedNodeOperator to a stack variable and use it since using SLOAD opcode twice consumes more gas than using it once and pushing the value to the stack for further usage.

5.3.35 Not Updated “slashed” Variable

SEVERITY	INFO
STATUS	FIXED

Description

#L288

The only place where slashed is updated is its initialization with 0. Later it is used in subtraction in line #L372.

Recommendation

To redesign logic of slashed usage

5.3.36 Not Updated “slashedTimestamp” Variable

SEVERITY	INFO
STATUS	FIXED

Description

#L289

The only place where slashedTimestamp is updated is its initialization with 0.

Recommendation

To redesign the logic of slashedTimestamp usage.

5.3.37 Nonoptimal Gas Consumption in getOperatorShares function

SEVERITY	INFO
STATUS	FIXED

Description

#L945

Gas consumption might be reduced in this line.

Recommendation

It might be better to place totalTimesValidatorsSlashed to a stack variable and use it since using sload opcode twice consumes more gas than using it once and pushing the value to stack for further usage.

5.3.38 Nonoptimal Gas Consumption in getOperatorRewardAddresses function

SEVERITY	INFO
STATUS	FIXED

Description

#L988
In the line gas consumption can be reduced.

Recommendation

It might be better to place slashingDelay to a stack variable and use it since using sload opcode twice consumes more gas than using it once and pushing the value to stack for further usage.

5.3.39 No Events Usage

SEVERITY	INFO
STATUS	DISMISSED

Description

LidoNFT doesn't emit any events.

Recommendation

To consider adding events for any important actions or user interactions for better logging and monitoring.

5.3.40 Inconsistent Naming of _newImplementation Variable

SEVERITY	INFO
STATUS	FIXED

Description

#L20

Recommendation

To rename _newImplementation to _implementation.

5.4 Revision 1 Summary

Issue Type	Number of Founded Issues
CRITICAL	4
MAJOR	0
WARNING	10
INFO	40
Total	**55**

6 Revision 2: Findings

6.1 CRITICAL

6.1.1 Possible Incorrect Contract State - “reservedFunds” Variable May Become Greater than “totalBuffered”

SEVERITY	CRITICAL
STATUS	NO ISSUE

Description

Using the issue from section 6.2.5 allowedAmount2RequestFromValidators will be changed by minValidatorBalance, which will be less than real. StMATIC.sol#L157

An attack example for StMATIC.requestWithdraw: there are 20 validators; all validators have minimal balance (10 Matic), minValidatorBalance manipulated to be 9:

- call requestWithdraw(uint256 _amount) with _amount = 50 - totalDelegated = 10*20 = 200
- totalBuffered = 0
- first require right side = 10 + 9*20 = 190, does not revert StMATIC.sol#L150-L156

```
require(
 (totalDelegated + totalBuffered) >=
     currentAmount2WithdrawInMatic +
         minValidatorBalance *
         operatorShares.length,
 "Too much to withdraw");
```

- allowedAmount2RequestFromValidators = 200 - 180 = 20 - 30
- 30 will be reserved even if there is not enough money on the contract.

Results:

1. `reservedFunds > totalBuffered`
2. delegate will revert on the first requirement for everyone, the claim will not work for this user

Using an attack above it is possible to manipulate convertStMaticToMatic, when reservedFunds set too big, getTotalPooledMatic will be less; Hence, balance in Matic will be bigger.

The attack example:

1. make `reservedFunds > totalBuffered`
2. `getTotalPooledMatic` will be < than it should

3. `convertStMaticToMatic` will return > than it should

Results:
1. `requestWithdraw` will allow withdrawing more than it should
2. submit will return less than it should

Moreover, it might be used to front-run and block a whale withdrawal:
1. An attacker sees a whale withdrawal
2. Calls `requestWithdrawal` using this bug in that way that when the attacker calls claim tokens in will decrease `totalBuffered` to 0 but leaves some in reservedFunds
3. Wait for `stakeManager.epoch() == usersRequest.requestTime`
4. An attacker calls `claimTokens`
5. A whale call will revert because there are not enough buffered to withdraw

Recommendation

To fix logic related with minValidatorBalance parameter to avoid manipulations possibilities

6.2 MAJOR

6.2.1 Incorrect Contract State in Case of Slashed Validator Becomes Unstaked

SEVERITY	MAJOR
STATUS	FIXED

Description

Slash may lead to unstake of StakeManager.sol#L703

Moreover, polygon governance may forceUnstake StakeManager.sol#L198

Another option: StakeManager.dethroneAndStake is called StakeManager.sol#L405

These cases are not handled properly.

Results:

1. The operator will not be able to exit until DAO calls stopOperator. NodeOperatorRegistry still marks it as active, however, a call to NodeOperatorRegistry.unstake will revert because it calls StakeManager.unstake StakeManager.sol#L417 that checks deactivationEpoch that is updated to non-zero value in StakeManager_unstake StakeManager.sol#L1111

2. Any call to StMATIC.delegate will fail because:

1. NodeOperatorRegistry.getOperatorInfos will return this operator as active => StMATIC.delegate will be called for this operator
2. StMATIC.buyVoucher for this operator will fail because
    1. It calls ValidatorShare.buyVoucher that calls ValidatorShare_buyShares ValidatorShare.sol#L115 that has onlyWhenUnlocked modifier ValidatorShare.sol#L374; lock is called in
        - StakeManager_unstake StakeManager.sol#L1118 that is called in:
            - StakeManager.forceUnstake StakeManager.sol#L198
            - StakeManager.slash StakeManager.sol#L703
            - StakeManager.dethroneAndStake StakeManager.sol#L405
            - StakeManager.unstake StakeManager.sol#L411
        - StakeManager_jail that is called in StakeManager.slash StakeManager.sol#L705

3. StMATIC.distributeRewards will revert because an operator does not generate any rewards, and stakeManager.withdrawRewards has require(rewards >= minAmount, "Too small rewards amount"); ValidatorShare.sol#L163

4. In StMATIC.requestWithdraw allowedAmount2RequestFromValidators will be less (because minValidatorBalance * operatorShares.length counts the unstaked operator) In the worst case, for example when all validators has minValidatorBalance + (minValidatorBalance / active operators count), it will be impossible to withdraw minValidatorBalance because of underflow in allowedAmount2RequestFromValidators calculation.

Recommendation

To handle a case when a validator slash leads to his/her unstake.

6.2.2 An Unstaked Operator may join

SEVERITY	MAJOR
STATUS	FIXED

Description

Call to addOperator NodeOperatorRegistry.sol#L209 and joinOperator do not check if an operator has a stake. So an unstaked operator may be added. Or they can be unstaked after they are added.

Results:

The same as in the issue from section 6.2.1.

Recommendation

To add check that an operator has a stake.

6.2.3 Incorrect Contract State in Case of Validator without validatorShare joins

SEVERITY	MAJOR
STATUS	FIXED

Description

Steps:
1. DAO adds a node operator without validatorShare
2. The node operator's owner call join
3. sm.getValidatorContract will return 0, no.validatorShare will be 0 (NodeOperatorRegistry.sol#L320)

Also StakeManager's governance can set validatorContract to a broken address (StakeManager.sol#L268)

Results:

1. stopOperator will fail for this node operator on IStMATIC(stMATIC).withdrawTotalDelegated(no.validatorShare);

NodeOperatorRegistry.sol#L267

Because StMATIC.sol#L415 will fail

2. unstake for the operator will fail for the same reason NodeOperatorRegistry.sol#L477
3. validatorShare2OperatorId will be incorrect, especially if 2 operators without validatorShare will join

   It will break exit/wait stats (see claimFee)
4. stMatic.requestWithdraw will fail because it calls getTotalStakeAcrossAllValidators which calls getTotalStake which calls _validatorShare.getTotalStake. However, _validatorShare might have 0 addresses or not a validatorShare address.

   delegate, claimTokens, withdraw TotalDelegated, distributeRewards and getTotalStakeAcrossAllValidators will also fail for similar reasons
5. stMatic.claimTokens won't work for the operator

This situation is unfixable without contracts redeployment.

Recommendation

To add a check that a joining validator has a validatorShare that != 0. To handle a case when the Polygon government will set validatorShare for a broken address.

6.2.4 Reentrancy in NodeOperatorRegistry.migrate

SEVERITY	MAJOR
STATUS	FIXED

Description

If there are >= 2 totalStoppedNodeOperator including an attacking one and reward address of an attacker will transfer an NFT back and then reenter migrate in onERC721Received it's possible to get invalid state, totalStoppedNodeOperator too small and totalWaitNodeOperator too big

Note that reward address can be changed anytime by an operator in setOperatorRewardAddress

Results:

1. Invalid getState that is impossible to fix after the attack
2. Another stopped operator will not be able to migrate
3. Events may be in incorrect order which may lead to errors in 3rd parties

Recommendation

To use CEI pattern or nonReentrant modifier from OpenZeppelin's ReentrancyGuard or similar.

6.2.5 minValidatorBalance Variable May be Manipulated

SEVERITY	MAJOR
STATUS	FIXED

Description

StMATIC.sol#L289

```
(uint256 validatorShare, ) = IValidatorShare(
    operatorShares[i].validatorShare
).getTotalStake(operatorShares[i].validatorShare);
```

getTotalStake is called for the validatorShare itself. It should not have any balance because any tokens sent to it will be locked.

Because of that minValidatorBalance may be manipulated by sending validator shares to itself

Results:

1. requestWithdraw will fail (because minValidatorBalance is always type(uint256).max) until one of validatorShare contract has some balance

2. After the manipulation requestWithdraw allow to withdrawal a validator to almost 0, it's riskier (slashing takes a bigger part of stake) and may lock distributeRewards for a long time, see 'Call to distributeRewards will revert if any validator has < 1 Matic in rewards'

3. An attacker may lock withdrawals temporary for this example:

    1. totalDelegated + totalBuffered = 1kk
    2. there are 20 validators, each has 50k delegated to it

Steps for a griefer:

1. From one validatorShare buy 500k of Matic

2. Send it to the validatorShare contract

3. minValidatorBalance will be 50k, because others are 0

4. requestWithdraw: (totalDelegated + totalBuffered) >= currentAmount2WithdrawInMatic + minValidatorBalance * operatorShares.length

    a. (totalDelegated + totalBuffered) = 1kk

    b. currentAmount2WithdrawInMatic = X

    c. minValidatorBalance * operatorShares.length = 50k * 20 = 1kk

Result:

all withdrawals are locked until someone sends any amount of shares to any other validatorShare contract.

Recommendation

To replace StMATIC.sol#L289 with getTotalStake(IValidatorShare(operatorShares[i].validatorShare))

6.2.6 A Slashed Validator May Lock delegate Method

SEVERITY	MAJOR
STATUS	FIXED

Description

When a validator gets slashed, it may become locked, so StMATIC.buyVoucher won't work (reverted because of onlyWhenUnlocked in _buyShares). StakeManager.sol#L705

StakeManager.sol#L1046

NodeOperatorRegistry.slashOperators does not account for that. So a malicious operator (or just a random one) may get slashed and paralyze delegation. In the case of a malicious operator, it can be fixed only by the DAO (which may take a lot of time depending on DAO implementation). And a slashed operator can't do anything, only unstake, unjail will not work because of require(validators[validatorId].status == Status.Locked, "Not jailed"); StakeManager.sol#L717

Recommendation

To fix NodeOperatorRegistry.unjail function

6.2.7 A Node Operator Signer May Lock delegate Function

SEVERITY	MAJOR
STATUS	FIXED

Description

Node operator signer may call StakeManager.updateValidatorDelegation and set validatorShare.delegation to false - => validatorShare._buyShares will revert on first require - => validatorShare.restake and validatorShare.buyVoucher will revert - => StMATIC.delegate will stop working until DAO do stop

Recommendation
To add code to handle cases when delegation is set to false.

6.2.8 Revert of distributeRewards Function In Case Some Validator Has Less Than 1 Matic in Rewards

SEVERITY	MAJOR
STATUS	FIXED

Description

Call to distributeRewards will fail if some validator has less than 1 Matic in rewards. See ValidatorShare.sol#L163 Because of that slashed operators may not get penalty because this function is always called after the penalty period has ended

Recommendation

To add handling of a case when an operator has < 1 Matic in rewards so distributeRewards won't fail and make sure it's called often enough. Possibly using a keeper.

6.2.9 A Slashed Operator may not Suffer any Reduction in Rewards

SEVERITY	MAJOR
STATUS	FIXED

Description

1. An operator gets slashed
2. NodeOperatorRegistry.slashOperators is called
3. NodeOperatorRegistry.slashingDelay has passed (~2.5h), StMATIC.distributeRewards is not called or reverts because of bug 'Call to distributeRewards will revert if any validator has < 1 Matic in rewards'.
4. slashOperators is called again, no.slashedTimestamp is set to 0 NodeOperatorRegistry.sol#L1037
5. distributeRewards is called, a slashed operator gets 100% of the rewards

Recommendation
To add handling of a case when an operator has < 1 Matic in rewards so distributeRewards won't fail and make sure it's called often enough. Possibly using a keeper.

6.3 WARNING

6.3.1 Impossibility of Restaking Rewards

SEVERITY	WARNING
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L455 _stakeRewards parameter for IValidator(no.validatorProxy).restake is always false

Recommendation
To replace false with _restakeRewards

6.3.2 Impossibility to Change Node Operator Status from “WAIT”

SEVERITY	WARNING
STATUS	FIXED

Description

Because exitOperator is never called, an operator with WAIT status will have this status forever. If there are too many operators with WAIT status ValidatorFactory will start to fail with ‘out of gas’ when setValidatorImplementation/setOperator is called.

Recommendation
To add a call to exitOperator

6.3.3 Failure of unjail Method in Case it is Called by an Unstaked but Unslashed Operator

SEVERITY	WARNING
STATUS	FIXED

Description

1. An operator calls unstake
2. An operator changes their mind and calls unjail
3. Call to unjail fails L520
   IValidator.unjail calls StakeManager.unjail which has require(validators[validatorId].status == Status.Locked, "Not jailed");

Status.Locked may be set only in _jail => slash

Recommendation

To update the docs or fix the unjail function

6.3.4 Incorrect Return Value of getValidatorStake Method for msg.sender

SEVERITY	WARNING
STATUS	FIXED

Description

Because of typo, == instead of =, at NodeOperatorRegistry.sol#L968 will return the balance of address(0) when address(0) is passed, not of msg.sender. It may lead to errors with 3rd parties.

Recommendation

To replace with single =.

6.3.5 A Slashed Operator Will Get Incorrect Share of Rewards

SEVERITY	WARNING
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L1090

```
uint256 t = _slashedTimestamp - block.timestamp;
uint256 penalty = ((t / slashingDelay) + (t == slashingDelay ? 0 : 1)) *
    10;
```

`t / slashingDelay` is 0 starting from the next block after the slash because both t and slashingDelay are units and `t < slashingDelay`.

That means that penalty = (0+1)*10=10 at the next block and (1+0)*10=10 at the same block the operator was slashed. But according to docs, the penalty should be 20.

Recommendation

To add precision constant for calculation using integer division or update the docs

6.3.6 The Risk of getApprovedTokens Getting out of Sync with getApproved

SEVERITY	WARNING
STATUS	FIXED

Description

In PoLidoNFT, if a user transfers to themself _tokenApprovals[tokenId] will be set to address(0), but `address2Approved` will still contain the token If someone will use getApprovedTokens as the source of truth they may think that they are approved, but they are not

Recommendation
To handle a case when from == to in PoLidoNFT._beforeTokenTransfer

6.3.7 entityFees are not in Sync with Documentation

SEVERITY	WARNING
STATUS	FIXED

Description

Docs say: “90% gets restaked, 5% goes to the treasury, 5% … among all active node operators.”

But actually it’s 2,5% to DAO, 2,5% to insurance and 5% to operators

entityFees = FeeDistribution(25, 50, 25); StMATIC.sol#L105

Recommendation

To update the docs or the code

6.3.8 Risk of Centralization

SEVERITY	WARNING
STATUS	ACKNOWLEDGED

Description

Centralization leads to a single point of failure. The initializer of NodeOperatorRegistry has all the rights to steal tokens from the contract. Or break the contracts.

The same goes for StMATIC and ValidatorFactory initializers.

Recommendation

To make sure it's a DAO or a multi-sig.

6.3.9 Possible Out of Gas if a lot of Operators Were Added and Removed

SEVERITY	WARNING
STATUS	ACKNOWLEDGED

Description

As times go by, validators are added and removed and the biggest operatorId may become so big that for-loop will run out of gas. NodeOperatorRegistry.sol#L988

Recommendation

To consider adding a way to request in bundles. Or rewrite in such a way that the operators array does not grow.

6.3.10 Potential No Penalties For Second Operator Slash

SEVERITY	WARNING
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L1029
1. the first slash happens
2. slashOperators is called
3. slashingDelay has passed, no.slashedTimestamp < block.timestamp
4. no one has called slashOperators
5. some more time has passed
6. The second slash happens
7. slashOperators is called

Depending on how much time step 5 will take a slashed operator may not receive any penalties for the second slash because no.slashedTimestamp + slashingDelay may be in the past

Recommendation

To set no.slashedTimestamp to block.timestamp + slashingDelay if no.slashedTimestamp is in the past.

6.3.11 Possible to DOS getApprovedTokens and getOwnedTokens

SEVERITY	WARNING
STATUS	ACKNOWLEDGED

Description

1. **owner2Tokens[someAddress]** size is never decreased. It's possible to increase the gas cost a lot for contracts that call getApprovedTokens for a certain user by calling approve a lot of times with 'to' set to a victim address May effectively DOS any calls to that function for that user (in another contract)
2. **address2Approved[someAddress]** size is never decreased. If an attacker receives a lot of empty/cheap tokens in StMatic he/she can then send it all to a victim and it may cause DOS in another contract that will read getOwnedTokens.

Recommendation

To add information about it to the docs or rewrite that an array size is decreased

6.3.12 Potential Unexpected Delegation Ratios

SEVERITY	WARNING
STATUS	FIXED

Description

StMATIC.sol#L269 Operators that were added earlier will get more in total than their ratio (maxDelegateLimit/maxDelegateLimitsSum) and operators with a high ratio that was added later will get less.

Recommendation

This behavior may be unexpected. A possible solution here would be to add that information to the docs or rewrite the logic.

6.3.13 Potential Too Many Rewards for Just Joined Operator

SEVERITY	WARNING
STATUS	ACKNOWLEDGED

Description

It may be significant depending on a period with which StMATIC.distributeRewards is called StMATIC.sol#L349

Recommendation

Make sure it's called often enough or don't add rewards for the first call to distributeRewards for new operators

6.3.14 Potential No Incentive to Call slashOperators Depending on Gas Cost

SEVERITY	WARNING
STATUS	FIXED

Description

Depending on the slash amount and gas cost, it may take a long time before it will make sense to call this function. Even for a system as a whole. E.g. calling it costs $1000 and slash was $300. There will be no motivation to call it even for a PoLido keeper. Any other operator has even less motivation because they get only a fraction of rewards from a slashed operator.

It's especially bad when a slash happens twice or more because only one slash will be recorded.

Recommendation

To consider rewriting the logic so no matter how often the function is called an operator will be slashed fairly.

6.4 INFO

6.4.1 "wait" Status Description is Wrong

SEVERITY	INFO
STATUS	FIXED

Description

Docs say WAIT: when the operator's owner claimed his tokens or migrated his NFT but the LidoMatic has not yet withdrawn the delegated Matics. In fact, the delegated Matics are withdrawn in stopOperator. NodeOperatorRegistry.sol#L477

Recommendation

To update the docs

6.4.2 Unreachable Statement

SEVERITY	INFO
STATUS	FIXED

Description

Condition in if is unreachable NodeOperatorRegistry.sol#L611

```
if (validatorShare2OperatorId[no.validatorShare] != 0)
```

It can be changed to 0 in exitOperator but exitOperator is never called. And we can't get here after deleting in else because of status EXIT

Recommendation
To add call to exitOperator

6.4.3 StMATIC does not extend IStMATIC interface

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry uses IStMATIC but StMATIC does not extend it.

Recommendation
To make StMATIC extend IStMATIC

6.4.4 User's claim Loss in Case of StakeManager.drainValidatorShares is Called by StakeManager Governance

SEVERITY	INFO
STATUS	FIXED

Description

1. User calls StMATIC.requestWithdraw
2. StakeManager.drainValidatorShares is called
3. User calls StMATIC.claimTokens and gets nothing

The same goes for withdrawTotalDelegated & claimTokens2StMatic

Recommendation
To reflect it in the docs

6.4.5 Wrong Description for rewardAddress

SEVERITY	INFO
STATUS	FIXED

Description

/// @param `rewardAddress` Validator public key used for access control and receive rewards.
NodeOperatorRegistry.sol#L34

Recommendation
To update the description

6.4.6 Unspecified Unit for maxDelegateLimit Parameter

SEVERITY	INFO
STATUS	FIXED

Description

/// @param maxDelegateLimit max delegation limit that StMatic contract will delegate to this operator each time delegate function is called.
NodeOperatorRegistry.sol#L43

Recommendation
Consider adding units used

6.4.7 Inconsistent naming of rewardAddress and operatorOwners variables

SEVERITY	INFO
STATUS	FIXED

Description

rewardAddress and operatorOwner are the same entity, they are updated and cleared in sync. It will be much easier to read and understand the code if the terms would be the same NodeOperatorRegistry.sol#L47, NodeOperatorRegistry.sol#L121

Recommendation

To consider renaming rewardAddress to operatorOwner or vice versa

6.4.8 Unconventional Modifier Name

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L164

Usually modifiers revert if a checked condition is false, e.g. userHasRole will revert if the user does not have the role. checkIfRewardAddressIsUsed reverts if the condition is true which is a little bit misleading. It may also be unexpected that this modifier checks for zero address.

Recommendation

To consider renaming to rewardAddressIsNotUsedNorZero/notUsedAsRewardAddressNorZero or similar

6.4.9 The Usage of non-self-explaining Modifier Names

SEVERITY	INFO
STATUS	DISMISSED

Description

It's not clear what some modifiers will do until its code or its comment is read, e.g. checkStakeAmount NodeOperatorRegistry.sol#L140, checkHeimdallFees, checkMaxDelegationLimit

Recommendation

Consider renaming checkStakeAmount to stakeAmountIsInAllowedBounds or similar, etc.

6.4.10 Missing Recommended Zero Address Checks in “initialize” Function and Setters

SEVERITY	INFO
STATUS	DISMISSED

Description

Zero address checks does not cost a lot of gas and may help in case of a mistake.

Recommendation
To consider adding zero address checks in initialize and setters

6.4.11 Non-use of “ether” Keyword for Units

SEVERITY	INFO
STATUS	DISMISSED

Description

It’s best to replace magic numbers with constant/keyword for readability and clearness of intentions NodeOperatorRegistry.sol#L187

```
minAmountStake = 10 * 10**18;
minHeimdallFees = 20 * 10**18;
```

Recommendation

To use ether instead of 10**18

6.4.12 The Usage of Magic Number Instead of Constant or Comment

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L189

slashingDelay = 2**13;

Recommendation

It would be better to replace magic numbers with constant or units (hours, minutes) for readability and clearness of intentions or at least add a comment

6.4.13 Different Levels of Abstraction in the Same Function

SEVERITY	INFO
STATUS	DISMISSED

Description

It's best for readability to keep a function at the same level of abstraction.

1. NodeOperatorRegistry.sol#L317

```
for (uint256 idx = 0; idx < operatorIds.length - 1; idx++) {
  if (_operatorId == operatorIds[idx]) {
      operatorIds[idx] = operatorIds[operatorIds.length - 1];
      break;
  }
}
operatorIds.pop();
```

2. NodeOperatorRegistry.sol#L356

```
address(uint160(uint256(keccak256(no.signerPubkey)))),
```

Recommendation

To consider moving low-level "get address from pub key" and "remove from an array" to a separate functions

6.4.14 Non-Use of Constant for getOperator

SEVERITY	INFO
STATUS	DISMISSED

Description

In getOperator(0) it's not 100% clear why 0 is used.

Recommendation
It might be more clear to use a constant,e.g. getOperator(OPERATOR_ID_FOR_MSG_SENDER)

6.4.15 Typos

SEVERITY	INFO
STATUS	FIXED

Description

1. NodeOperatorRegistry.sol#L468 - "quite" instead of "quit"

```
/// @dev when the operators's owner wants to quite the PoLido protocol he can call
```

2. NodeOperatorRegistry.sol#L1013 - "slahed" instead of "slashed"

```
checkCondition(length == operatorIds.length, "slahed operators length");
```

Recommendation
To fix typos

6.4.16 An Operator's Exit without a Stop from DAO Leads to Unstaking and Losing a Slot

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L476
It may be inconvenient for a validator to join if an exit costs them a slot.

Recommendation
To recheck that it's a desired logic or change it

6.4.17 Ambiguity in setCommissionRate natspec

SEVERITY	INFO
STATUS	DISMISSED

Description

/// @notice Allows to set the commission rate used.
NodeOperatorRegistry.sol#L741
It's not clear that the commission rate will only be used for new operators

Recommendation
To add a clarification that it will set a new commission rate only for new validators, old ones will keep an old commission rate

6.4.18 Unnecessary Storage Read

SEVERITY	INFO
STATUS	DISMISSED

Description
NodeOperatorRegistry.sol#L1035

You can save gas by reading from memory, slashedTimestamp

```
else if (
    slashedTimestamp != 0 &&
    no.slashedTimestamp < block.timestamp
)
```

Recommendation
To replace `no.slashedTimestamp` with slashedTimestamp

6.4.19 Uninitialized Local Variables

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L1058, NodeOperatorRegistry.sol#L884

uint256 index;
StMATIC.sol#L147

unit256 allowedAmount2RequestFromValidators;

Recommendation

To explicitly set them to zero in order to improve code readability. To use slither to detect them.

6.4.20 checkCondition is Redundant

SEVERITY	INFO
STATUS	DISMISSED

Description

NodeOperatorRegistry.sol#L1100. It will be easier to read code with requires than with checkConditions

Recommendation
To replace checkCondition with require

6.4.21 Missing Checks-Effects-Interactions pattern

SEVERITY	INFO
STATUS	DISMISSED

Description

It's a best practice to always use CEI. Several places don't use it. Not all of them lead to reentrancy now, but this may lead to in the future when some of your or polygon contracts are changed. It can be found with slither.

Recommendation
To consider always using CEI pattern

6.4.22 PoLidoNFT Does Not Extend IPoLidoNFT interface

SEVERITY	INFO
STATUS	FIXED

Description

PoLidoNFT does not extend IPoLidoNFT

Recommendation
Add IPoLidoNFT to parents of IPoLidoNFT

6.4.23 PoLidoNFT.tokenId2ApprovedIndex Does Not Include Ones From setApprovalForAll

SEVERITY	INFO
STATUS	FIXED

Description

It may be unexpected because there is no information about it in the documentation.

Recommendation
To add the information about this topic to the docs.

6.4.24 Non external visibility for PoLidoNFT.initialize, StMATIC.initialize and ValidatorFactory.initialize

SEVERITY	INFO
STATUS	FIXED

Description

To save gas and to explicitly show that it will not be called inside the contract

Recommendation
To set visibility to external

6.4.25 “_name” and “_symbol” Shadow Ones From ERC721Upgradeable

SEVERITY	INFO
STATUS	FIXED

Description

PoLidoNFT.sol#L35 Best practice is to not shadow existing variables.

Recommendation

To rename shadow variables ‘name’ and ‘symbol’. To use slither to detect shadowing.

6.4.26 No Token Existence Checks in PoLidoNFT.burn function

SEVERITY	INFO
STATUS	FIXED

Description

PoLidoNFT.sol#L67

There are no checks that given tokenId exists. Right now it’s impossible to call it because of checks in StMATIC, but it may change in the future and it’s better to check arguments inside burn function. Burning non-existing token will lead to Minting path inside _beforeTokenTransfer.

Recommendation
To check that a _tokenId exists inside burn

6.4.27 Misleading Naming of “lastApprovedTokens” Variable

SEVERITY	INFO
STATUS	FIXED

Description

PoLidoNFT.sol#L198

Recommendation
To rename the variable to approvedTokens

6.4.28 Redundant Writing to storage of approvedTokens for address(0)

SEVERITY	INFO
STATUS	FIXED

Description

PoLidoNFT.sol#L84

You may save a little bit of gas by not writing for 0 address and it will also help to keep storage clean from unused data

Recommendation

To check for address(0) and do not write to storage for it

6.4.29 Potential Gaps and Incorrect Length of PoLidoNFT.getOwnedTokens and getApprovedTokens Arrays

SEVERITY	INFO
STATUS	DISMISSED

Description

Array that is returned from `getOwnedTokens/getApprovedTokens` may have gaps (0s), length

may be incorrect (e.g. when someone calls `approve/removeApproval` or burn a token)

Recommendation
To make sure it's accounted for in front-end and is described for 3rd parties. Or to rewrite in order to decrease the length of the arrays.

6.4.30 Non external visibility for PoLidoNFT.getOwnedTokens and getApprovedTokens functions

SEVERITY	INFO
STATUS	FIXED

Description

PoLidoNFT.sol#L172, PoLidoNFT.sol#L185

To save gas and to explicitly show that it will not be called inside the PoLidoNFT those functions can be declared as external.

Recommendation
To declare these functions with external visibility

6.4.31 Misleading Naming of "requestTime" Variable

SEVERITY	INFO
STATUS	FIXED

Description

It stores epoch, not time.

StMATIC.sol#L64

Recommendation

To rename to requestEpoch

6.4.32 Misleading Naming of “nodeOperator” Variable

SEVERITY	INFO
STATUS	FIXED

Description

It stores node operator registry, not just an operator

StMATIC.sol#L97

Recommendation

To rename to nodeOperatorRegistry

6.4.33 Incorrect IStakeManager.withdrawRewards Return Type

SEVERITY	INFO
STATUS	FIXED

Description

IStakeManager.sol#L57

```
function withdrawRewards(uint256 validatorId) external returns (uint256);
```

StakeManager.sol#L516

```
function withdrawRewards(uint256 validatorId) public onlyStaker(validatorId) {
```

Recommendation

To remove returns (uint256)

6.4.34 DOS in Case of the Delegation is Disabled by Stakemanager

SEVERITY	INFO
STATUS	FIXED

Description

if StakeManager.delegationEnabled is false a user will be able to call StMATIC.submit but not withdraw and almost all the other functions (restake, requestWithdraw, delegate, withdrawTotalDelegated) will revert.

Recommendation
To make sure it's highly unlikely that delegationEnabled will be false. One more possible option is to add information to the docs

6.4.35 Redundant Variable "totalAmount2WithdrawInMatic"

SEVERITY	INFO
STATUS	FIXED

Description

StMATIC.sol#L142

```
uint256 totalAmount2WithdrawInMatic = convertStMaticToMatic(_amount);
uint256 currentAmount2WithdrawInMatic = totalAmount2WithdrawInMatic;
```
may be replaced with

```
uint256 currentAmount2WithdrawInMatic = convertStMaticToMatic(_amount);
```

Recommendation

To remove redundant intermedia variable.

6.4.36 Out of Gas Exception in StMATIC, ValidatorFactory in Case of too Many Validators are Added

SEVERITY	INFO
STATUS	DISMISSED

Description

It was discussed earlier that there should not be more than 20 however, there are no such limits in code.

Recommendation
To consider limiting validators in code

6.4.37 Magic Number for minValidatorBalance Calculation

SEVERITY	INFO
STATUS	DISMISSED

Description

StMATIC.sol#L292

It's not clear in the code why *10/100 is used. It may be replaced with /10. It may also be extracted to a variable or constant.

Recommendation

To replace the magic variable with a constant or a named variable

6.4.38 Unnecessary Multiplication by 1

SEVERITY	INFO
STATUS	FIXED

Description

StMATIC.sol#L359
Multiplication by 1 does not change anything.

Recommendation
To remove unnecessary operation

6.4.39 Unnecessary Complexity in Calculation

SEVERITY	INFO
STATUS	FIXED

Description

StMATIC.sol#L400

totalBuffered += (currentBalance - totalBuffered);
Might be replaced with totalBuffered = currentBalance

Recommendation
To remove unnecessary complexity.

6.4.40 Incorrect Term in natspec

SEVERITY	INFO
STATUS	FIXED

Description

StMATIC.sol#L406

* @notice Only NodeOperator can call this function

Recommendation

Replace NodeOperator with NodeOperatorRegistry

6.4.41 Nonoptimal Gas Usage in claimTokens2StMatic and claimTokens Functions

SEVERITY	INFO
STATUS	FIXED

Description

It won't be used anymore so it can be deleted.

StMATIC.sol#L441, StMATIC.sol#L310

Recommendation

To delete lidoRequest/usersRequest at the end of claimTokens2StMatic/claimTokens function.

6.4.42 StMATIC.restake Function is Unused

SEVERITY	INFO
STATUS	FIXED

Description

StMATIC.sol#L511

Recommendation
To remove unused function

6.4.43 Unexpected Result for a User in Case of Withdrawn Value is Less than it was Requested

SEVERITY	INFO
STATUS	FIXED

Description

ValidatorShare.unstakeClaimTokens_new use the same variables for all withdrawals
ValidatorShare.sol#L87

```
function withdrawExchangeRate() public view returns (uint256) {
      ...
      return _withdrawShares == 0 ? precision :
withdrawPool.mul(precision).div(_withdrawShares);
  }
```

ValidatorShare.sol#L305

```
withdrawShares = withdrawShares.sub(shares);
withdrawPool = withdrawPool.sub(_amount);
```

Thus, if a user requests a withdrawal, then a slash happens, then someone else requests a withdrawal a user will receive less than he/she requested

Recommendation
To add that information to the docs

6.4.44 Comments Without Meaning

SEVERITY	INFO
STATUS	FIXED

Description

Validator.sol#L123

// stakeManager
Validator.sol#L181

// polygonERC20
// stakeManager

Recommendation

To remove useless comments.

6.4.45 Non-Needed Approval

SEVERITY	INFO
STATUS	FIXED

Description

Approval is not used and is cleared by calling safeTransferFrom

Validator.sol#L257

```
erc721.approve(_rewardAddress, _validatorId);
erc721.safeTransferFrom(address(this), _rewardAddress, _validatorId);
```

Recommendation

To remove erc721.approve(_rewardAddress, _validatorId);

6.4.46 Lack of Information Regarding the Obligatory Implementation of EIRC721Receiver for rewardAddress Contract in Case of Migration

SEVERITY	INFO
STATUS	ACKNOWLEDGED

Description

safeTransferFrom will fail if _rewardAddress is not IERC721Receiver.

```
erc721.safeTransferFrom(address(this), _rewardAddress, _validatorId);
```

Validator.sol#L258

Recommendation

To add that information to the docs

6.4.47 Misleading Use of onERC721Received

SEVERITY	INFO
STATUS	DISMISSED

Description

Validator.sol#L287 onERC721Received should be removed because Validator.sol doesn't know how to handle other NFTs and you don't want them to be sent to the contract. Someone may accidentally call safeTransferFrom function with the parameter to equal to this contact address, and an NFT will be stuck. It will also help to save on deployment cost.

Validator.sol#L276 After that replace safeTransferFrom with transferFrom in
erc721.safeTransferFrom(_rewardAddress, address(this), _validatorId);

Recommendation

To remove onERC721Received and use transferFrom

6.4.48 Redundant require Statement

SEVERITY	INFO
STATUS	FIXED

Description

No need for this check because isOperator modifier implicitly checks it and this error will never be triggered

```
require(operator != address(0), "Operator contract not set");
```

ValidatorFactory.sol#L41

Recommendation

To remove redundant require

6.4.49 Usage of += Operator instead of =

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L418

```
no.amountStaked += _amount;
```

An operator can call stake only after addOperator is called where amountStaked is set to 0

Recommendation
To replace += with =

6.4.50 Non-Use of Events for Critical Setters

SEVERITY	INFO
STATUS	FIXED

Description

E.g. NodeOperatorRegistry.setRestake, setUnjail, setStMATIC, setValidatorFactory, etc.

Recommendation
To add events for critical setters so your user and you can track those changes

6.4.51 No Verification for Zero Amount in Function Input

SEVERITY	INFO
STATUS	FIXED

Description

At the line StMATIC.sol#L137 function requestWithdraw(uint256) has _amount argument which has no verification for not being zero. If it's zero, the function will be executed for no reason

Recommendation
To verify that _amount argument is not zero

6.4.52 Unnecessary State Write

SEVERITY	INFO
STATUS	FIXED

Description

At the line StMATIC.sol#L420 if there is no amount to be withdrawn, the function still mints poLidoNFT and consumes gas.

Recommendation

To replace return with revert.

6.4.53 Usage of array.length in Loops

SEVERITY	INFO
STATUS	FIXED

Description

At the lines: - StMATIC.sol#L250 - StMATIC.sol#L288 - StMATIC.sol#L353 - StMATIC.sol#L380 - StMATIC.sol#L386 - StMATIC.sol#L586

loops use array.length as high bound, which consumes more gas than it's separation to variable and using this variable

Recommendation

To create a stack variable to store an array.length value and use it as a loop bound.

6.4.54 Multiple Consecutive Calls

SEVERITY	INFO
STATUS	FIXED

Description
Starting at the line Validator.sol#L64 there are 3-4 consecutive calls to stakeManager.

Recommendation

To save gas, we would recommend changing/expanding the interface of IStakeManager by adding a method that aggregates the functionality of these 3-4 methods.

6.4.55 The Difference of totalPooledMatic Formula in Different Parts of Docs

SEVERITY	INFO
STATUS	FIXED

Description

Lido for Polygon Architecture Docs
StMATIC Docs

Recommendation

To update the docs

6.5 Revision 2 Summary

Issue Type	Number of Founded Issues
CRITICAL	**0**
MAJOR	9
WARNING	14
INFO	55
Total	**78**

7 Revision 3: Findings

7.1 WARNING

7.1.1 Possible Front-Run of setStMATIC, setFxRootTunnel, setFxChildTunnel

SEVERITY	WARNING
STATUS	ACKNOWLEDGED

Description

setStMATIC is not protected, anyone can call it. It's possible to front-run a deployer. This will require redeployment because it will lead to DoS of StMatic. It's better to set stMATIC in the constructor.

FxStateRootTunnel.sol#L21

Because there is no zero address check it's possible to set stMatic to address(0) by mistake which will also allow an attacker to DoS StMatic. Moreover, setFxRootTunnel and setFxChildTunnel are not protected from the same attack.

Recommendation

To set stMATIC in the constructor. Consider calling setFxRootTunnel, setFxChildTunnel in the constructor.

7.2 INFO

7.2.1 The Rule Of FxStateChildTunnel.getReserves Consideration

SEVERITY	INFO
STATUS	ACKNOWLEDGED

Description

It should be kept in mind that StMatic may become out of sync with data sent via fx-portal and it's quite easy to manipulate that by depositing/withdrawing funds. Also adding/removing a validator,

changing its state, slashing, calling distributeRewards, etc. will change balances but will not trigger events and leads to out-of-sync data.

Recommendation

To add a note in the documentation that FxStateChildTunnel.getReserves shouldn't be considered as a reliable source of data.

7.2.2 Unnecessary Use of Assembly

SEVERITY	INFO
STATUS	DISMISSED

Description

Assembly significantly decreases code readability. It's error-prone and it's better to avoid it when possible.

FxStateChildTunnel.sol#L34

Recommendation

To use abi.decode(latestData, (uint256, uint256)).

7.2.3 Commented Code

SEVERITY	INFO
STATUS	FIXED

Description

FxStateChildTunnel.sol#L41

Unused code should be removed because it decreases readability.

Recommendation

To remove commented code.

7.2.4 The Option of Copy-Paste Code Replacement With npm Package

SEVERITY	INFO
STATUS	DISMISSED

Description

Contracts added to lib and tunnel directories may be imported from [@maticnetwork/fx-portal](https://www.npmjs.com/package/@maticnetwork/fx-portal) package. It will be more readable for end-users and auditors because they will not have to verify that you have not changed the copied contracts. It will also help to separate your code from the external one.

Recommendation

To remove copy-pasted contracts and use ones from npm package.

7.2.5 The Risk of Becoming Out-of-Sync by StMatic Between the Contracts

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry and PoLidoNFT have setStMatic functions. PoLidoNFT and FxStateRootTunnel has stMATIC getters. That leads to several sources of truth. It may lead to some errors in 3rd parties.

FxStateRootTunnel doesn't allow to change StMatic, it means that it's necessary to redeploy FxStateRootTunnel contract every time stMatic is changed.

Recommendation

When updating StMatic make sure that the update of it's values is not forgotten at all 3 contracts or to rewrite them in order to have the single source of truth.

7.2.6 Immutable “stMatic” variable in FxStateRootTunnel Contract

SEVERITY	INFO
STATUS	FIXED

Description

Other contracts allow changing stMatic address by the setter. But FxStateRootTunnel is immutable and will have to be redeployed which costs more gas than a setter. It will also require redeploying FxBaseChildTunnel. It will lead to state reset and stale data on old FxBaseChildTunnel which may lead to errors in 3rd parties.

Recommendation
To consider adding a setter and adding information above to the docs.

7.2.7 Non-Fixed Pragma Version

SEVERITY	INFO
STATUS	DISMISSED

Description

It is considered best practice to pick one compiler version and stick with it because different compilers have different bugs and peculiarities. FxStateRootTunnel.sol#L2 FxStateChildTunnel.sol#L2
pragma solidity ^0.8.0;

Recommendation

To use a fixed version as used in other contracts.
pragma solidity 0.8.7

7.2.8 Non-Use of IFxStateChildTunnel

SEVERITY	INFO
STATUS	DISMISSED

Description

IFxStateChildTunnel is not used.

Recommendation
To remove unused code.

7.3 Revision 3 Summary

Issue Type	Number of Founded Issues
CRITICAL	0
MAJOR	0
WARNING	1
INFO	8
Total	**9**

8 Revision 4 Findings

8.1 CRITICAL

8.1.1 Possible Incorrect Exchange Rate from stMatic to Matic

SEVERITY	CRITICAL
STATUS	FIXED

Description

It can be manipulated by a validator that becomes jailed deliberately. Or an attacker may wait for a validator to become jailed and front-run the transaction that will set the validator status to jailed.
Example: 2 validators, 100 Matic staked at each, 200 stMatic minted, 1 stMatic = 1 Matic
An attacker waits for a validator to be jailed
The attacker calls submit(100), sent 100 Matic, gets 200 stMatic
An attacker waits for a validator to be unjailed
Attacker exchanges 200 stMatic for 200*3/4=150 matic

a. totalShares = 200
b. totalPooledMATIC = 100
c. exchange rate will become 100/200 => 2 stMatic = 1 Matic
a. totalShares = 400
b. totalPooledMATIC = 300
c. exchange rate is 300/400 => 3 stMatic = 4 maticResults:

50 Matic profit for the attacker, loss for every other participant
a validator may use it to front-run a whale, so the whale withdrawal will return less Matic
fxStateRootTunnel.sendMessageToChild will send unexpected values

Recommendation

To rewrite the logic that calculates stMatic to Matic rate

8.1.2 Lock of Delegated Money in Case of an Operator is Unstaked During The Slash

SEVERITY	CRITICAL
STATUS	FIXED

Description

An operator may be unstaked during slash
StakeManager.sol#L703
if (validatorStakeSlashed == 0) { _unstake(validatorId, currentEpoch); }
Delegated money will be locked forever because:
requestWithdraw allows to withdraw only from active(inactive are filtered in getOperatorInfos), but it has status 'UNSTAKED'
unstake will also revert because of getOperatorStatus(no) == NodeOperatorStatus.ACTIVE
It will have status UNSTAKED forever, no way to change it
stopOperator will revert because of status <= NodeOperatorStatus.ACTIVE
unstake will revert because of
getOperatorStatus(no) == NodeOperatorStatus.ACTIVE
exitOperator will never be called
=> because claimTokens2StMatic will never be called with tokenId for this operator
=> because withdrawTotalDelegated will never be called for this validatorShare
=> because unstake will revert
Almost the same goes for jailed, but jailed validator may call unjail and fix the issue. It gives him/her the ability to:

- manipulate stMatic/Matic exchange rate when he/she wants it.
- go to jail before a big withdrawal that will lower totalDelegated and the withdrawal will revert
- lock all the money delegated to him/her and then ask for a ransom

Recommendation

To rewrite the logic to make sure that the validator share of validators having JAILED or UNSTAKED status are stoppable and unstakable

8.1.3 No Operator May Join or Stake

SEVERITY	CRITICAL
STATUS	FIXED

Description

joinOperator has a check that an operator status is INACTIVE
getOperator returns a nodeOperator with validatorId set to 0 (set in addOperator)
getOperatorStatus will request IStakeManager.Validator with that validatorId
returned status will be incorrect, a validator with id 0 may have a status other than INACTIVE
joinOperator will revert
NodeOperatorRegistry.sol#L304
The same goes for stake

Recommendation

To rewrite the logic so the status of joining/staking operator does not depend on validator with ID 0.

8.1.4 Failure of “delegate” Function on a Second Call

SEVERITY	CRITICAL
STATUS	FIXED

Description

StMATIC.sol#L283
Not everything will be distributed after the first call, some approval will remain.
Because of that second call to safeApprove will fail

Recommendation

To call safeApprove(address(stakeManager),0) before calling safeApprove(address(stakeManager), totalToDelegatedAmount)

8.2 MAJOR

8.2.1 No Incentive to Join for New Operators

SEVERITY	MAJOR
STATUS	DISMISSED

Description

Because of the issue 8.3.2 below there will be less and less incentive for new validators to join because all the rewards will go to old ones.

Recommendation

To change getOperatorInfos(true) to getOperatorInfos(false)

8.3 WARNING

8.3.1 Unchangeable WAIT status

SEVERITY	WARNING
STATUS	FIXED

Description

WAIT status can only be changed in

```
exitOperator
```

exitOperator is called inside
claimTokens2StMatic , which must be called after

```
withdrawTotalDelegated
```

Which is called by:

- stopOperator , that requires status INACTIVE or ACTIVE
- unstake , that requires status ACTIVE

Thus, the operator with WAIT status will always stay WAIT.

Recommendation

To rewrite the logic, so it's possible to change WAIT status.

8.3.2 Unfair Rewards Distribution for New Operators

SEVERITY	WARNING
STATUS	FIXED

Description

Validators with an already big stake (old ones) will have rewards > threshold more often
=> more often they will be returned by getOperatorInfos
=> more stake leads to more delegation, more delegation to more stake
=> early ones will become bigger and bigger while the new ones won't get as much
It especially dangerous because of If an operator was unstaked during slash delegated money will be locked forever

Recommendation

To change getOperatorInfos(true) to getOperatorInfos(false)

8.3.3 Potential Delegation Disablement using Front Running

SEVERITY	WARNING
STATUS	FIXED

Description

A griefer can disable delegation by front-running calls to delegate and calling

```
distributeRewards
```

Or just call
distributeRewards often enough

Recommendation

To change getOperatorInfos(true) to getOperatorInfos(false)

8.3.4 Potential Overstepping of “maxDelegateLimitsSum” variable

SEVERITY	WARNING
STATUS	ACKNOWLEDGED

Description

StMATIC.sol#L273
Because of checks to
if (validator.delegation())
Prerequisites:
one of validators has delegation set to false
maxDelegateLimitsSum <= availableAmountToDelegate
totalToDelegatedAmount = maxDelegateLimitsSum
amountToDelegatePerOperator is less than required for full delegation

Result: not everything is delegated because denominator is bigger than it should be in

```
(operatorShares[i].maxDelegateLimit * totalToDelegatedAmount) / maxDelegateLimitsSum
```

Recommendation

To consider not adding maxDelegateLimit to maxDelegateLimitsSum for operators with delegation set to false.

8.3.5 Incorrect Contract State in case of the Polygon Government set validatorShare to a Invalid Address

SEVERITY	WARNING
STATUS	ACKNOWLEDGED

Description

See issue 6.2.3 in the Revision 2 report. Also issues 6.3.1, 6.3.2, 6.3.4 and 6.3.5 is not fixed

Recommendation

To handle a case when the Polygon government will set validatorShare for a broken or zero address.

8.4 INFO

8.4.1 Non-plural “totalNodeOperator” Variable Naming

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L68
uint256 private totalNodeOperator;

Recommendation

To change to

```
totalNodeOperators
```

8.4.2 Misleading Status Calculation at Runtime for INACTIVE and UNSTAKED Statuses

SEVERITY	INFO
STATUS	DISMISSED

Description

When someone reads that code they expect that the status will be set to INACTIVE/UNSTAKED.
NodeOperatorRegistry.sol#L204

```
operators[operatorId] = NodeOperator({ status: NodeOperatorStatus.INACTIVE,
```

NodeOperatorRegistry.sol#L438

```
no.status = NodeOperatorStatus.UNSTAKED;
```

However, in fact, it will be ignored and will be calculated at runtime from stakeManager in getOperatorStatus .

Recommendation

To consider adding a status that shows that it will be calculated at runtime. E.g. CALCULATE_AT_RUNTIME, REQUEST_FROM_STAKE_MANAGER

8.4.3 Commented Code

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L265
// no.statusUpdatedTimestamp = block.timestamp;

Recommendation

To remove this commented code code.

8.4.4 Misleading Function Argument

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L398

_restakeRewards argument in restake function is not used.
It may be misleading for a user, they will expect that restakeRewards will restake rewards as the natspec says.
Moreover, if it's called with 0 amount nothing will change in the state but user may expect that they have restaked.

Recommendation

To remove _restakeRewards argument

8.4.5 Incorrect NatSpec for unjail Function

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L464

/// @dev when an operator is UNSTAKED the owner can switch back and stake the
/// operator by calling the unjail func, in this case, the operator status is set
/// to back ACTIVE.

Recommendation

To replace UNSTAKED with JAILED

8.4.6 Meaningless Variable Names

SEVERITY	INFO
STATUS	FIXED

Description

It would be better for readability to use full names

```
_op => nodeOperator, res => result, v => validator
```

NodeOperatorRegistry.sol#L799
length => operatorIdsLength
NodeOperatorRegistry.sol#L901

```
uint256 length = operatorIds.length;
```

Recommendation

To consider using meaningful variable names

8.4.7 Incongruity in the PR Description

SEVERITY	INFO
STATUS	DISMISSED

Description

You wrote in PR that “Handled by getting the current state of validator in functions getOperatorInfos and getOperatorIds directly from the StakeManager.”
However, in fact, getOperatorIds is returned from the contract storage.

Recommendation

To make sure that it’s the expected behavior

8.4.8 Misleading Naming of getOperatorInfos Function

SEVERITY	INFO
STATUS	DISMISSED

Description

A user may expect that they will get info about all the operators, however, in fact,they get it only for active ones.
_rewardData may also be misleading because you don’t expect it to filter ones without reward

Recommendation

To rename those functions to getActiveOperatorInfos, _ignoreOnesWithoutRewards or similar

8.4.9 High Cyclomatic Complexity

SEVERITY	INFO
STATUS	DISMISSED

Description

High cyclomatic complexity decreases readability.
for->if->if->if is too deep
NodeOperatorRegistry.sol#L967

Recommendation

To consider splitting it into several functions

8.4.10 Typo “herimdall”

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L356
/// @param _heimdallFee herimdall fees.

Recommendation

To replace herimdall with heimdall

8.4.11 Misleading Naming of operatorShares Variable

SEVERITY	INFO
STATUS	FIXED

Description

operatorShares should be renamed to

```
operatorInfos
```

StMATIC.sol#L144
StMATIC.sol#L258
StMATIC.sol#L636

validatorShare to

```
stakedAmount
```

StMATIC.sol#L317

RecommendationTo rename the variables

8.4.12 Unreachable Statement

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L548
validatorShare2OperatorId[no.validatorShare] is never 0 here
It is set to 0:
after exitOperator is called
Impossible to call claimFee , status is set to EXIT
after claimFee is called
Impossible to call it again, status is set to EXIT

Recommendation

To remove unreachable statement

8.4.13 Debug Import

SEVERITY	INFO
STATUS	FIXED

Description

In Validator and StakeManagerMock

Recommendation

To remove debug import

8.5 Revision 4 Summary

Issue Type	Number of Founded Issues
CRITICAL	4
MAJOR	1
WARNING	5
INFO	13
Total	**23**

9 Revision 5: Findings

9.1 CRITICAL

9.1.1 Possible Funds Lock for Unstaked Operator

SEVERITY	CRITICAL
STATUS	FIXED

Description

An unstaked operator may call *claim* and then *claimFee* and get status 'EXIT'

Steps:

1. Evil operator calls unstake or is unstaked by the DAO
    - StMatic.withdrawTotalDelegated is called, funds are locked until claimTokens2StMatic is called, and we need to wait for a withdrawalDelay to call it
    - The operator becomes UNSTAKED
2. The operator waits for withdrawalDelay, and front-run call to claimTokens2StMatic
    - It may be very easy and cheap because claimTokens2StMatic is not probably called the same block the request epoch is reached
    - But even if it is the attacker may just overbid the call or use a private relayer
3. He/she calls unstakeClaim , becomes CLAIMED
4. He/she calls claimFee , becomes EXIT (WAIT is impossible because of '12. Unreachable statement' in last report)
5. claimTokens2StMatic fails inside a call to nodeOperatorRegistry.exitOperator
   StMATIC.sol#L498
   NodeOperatorRegistry.sol#L261

Result: Funds that were delegated to this operator are locked forever.

Recommendation

To add logic to handle this case.

9.1.2 Potential Non Return of Delegated Funds for Jailed Operators

SEVERITY	CRITICAL
STATUS	FIXED

Description

See the issue from section 8.1.2
StMATIC.requestWithdraw does not allow withdrawing from a jailed one

NodeOperatorRegistry.stopOperator does not allow to stop it and withdrawTotalDelegated
The status may be changed only by the validator by calling unjail
The attacker may lock the funds and ask for a ransom.

Recommendation

To allow withdrawing from a jailed validator

9.1.3 Potential Non Return of Delegated Funds for Ejected Operators

SEVERITY	CRITICAL
STATUS	FIXED

Description

The issue from section 9.1.2 is relevant here as well.
StMATIC.requestWithdraw does not allow withdrawing from an ejected one
NodeOperatorRegistry.stopOperator does not allow to stop it and withdrawTotalDelegated
The status may be changed only by unstake, but because of 'An unstaked operator may lock funds delegated to it' it may not help
The attacker may lock the funds and ask for a ransom.

Recommendation

To allow withdrawing from an ejected validator, add logic to handle 'An unstaked operator may lock funds delegated to it'

9.2 MAJOR

9.2.1 Unexpected INACTIVE Status for Validator Whos is Jailed and Unstaked Same Time

SEVERITY	MAJOR
STATUS	FIXED

Description

A jailed operator may become unstaked.
It will lead to deactivationEpoch != 0 and to status INACTIVE
NodeOperatorRegistry.sol#L849
Results:
join will fail because of check poValidator.status == IStakeManager.Status.Active
NodeOperatorRegistry.sol#L320
stake will create a new validatorShare contract, so delegated funds will be lost
may only be stopped (stopOperator) by the DAO which will lead to loss of delegated funds

Recommendation

To handle a case when an operator is jailed and unstaked

9.3 WARNING

No warning issues were found

9.4 INFO

9.4.1 Missing punishment for Disabling Delegation

SEVERITY	INFO
STATUS	FIXED

Description

A validator may disable delegation, and it will not be punished for it, he/she will get the rewards for funds already delegated to him/her

Recommendation

To consider adding a punishment because it may lead to uneven delegation

9.4.2 Misleading Naming of “NodeOperator.statusUpdatedTimestamp” variable

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L835
statusUpdatedTimestamp can not be updated for statuses that are calculated at runtime

Recommendation

To add that info to the docs or remove it if it’s not used

9.4.3 Uneven Rewards Distribution Depending on Delegated Funds Value

SEVERITY	INFO
STATUS	DISMISSED

Description

Because of a filter in getOperatorInfos(true, false, false) most rewards will go to the validators that have most funds delegated to them which is not fair.
It's also possible to front-run *distributeRewards* just before the small one should receive rewards. And then just after that. Because of that small validator may receive almost nothing

Recommendation

To consider updating distributeRewards logic.

9.4.4 Ambiguous Term "Active"

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L971
It is better when all used terms have an same meaning. NodeOperatorStatus.ACTIVE and *_allActive* parameter uses the same word "active" but in different meanings.

Recommendation

To consider using a separate term for ACTIVE, EJECTED, JAILED. E.g. `_allWithStake`

9.4.5 Flags in One Function Lower Readability

SEVERITY	INFO
STATUS	DISMISSED

Description

NodeOperatorRegistry.sol#L974
function getOperatorInfos(bool _withdrawRewards, bool _delegation, bool _allActive
Because of 3 flags a call to getOperatorInfos may look like

getOperatorInfos(true, true, false). It would be more readable if it would use packed flags, e.g.

getOperatorInfos(InfoFlags.WTIHDRAW_REWARDS & InfoFlags.DELEGATION) (bitwise and).

Recommendation

To consider using packed flags or a struct

9.4.6 Incongruity of Flags and Filters

SEVERITY	INFO
STATUS	DISMISSED

Description

NodeOperatorRegistry.sol#L974
function getOperatorInfos(bool _withdrawRewards, bool _delegation, bool _allActive
It would be a little more readable if filters and flags were in the same order. I.e.

```
if (_withdrawRewards) { come first
```

inside the for loop or
_withdrawRewards flag is last.

Recommendation

To consider reordering the flags or the filters

9.4.7 Complex Conditional

SEVERITY	INFO
STATUS	DISMISSED

Description

NodeOperatorRegistry.sol#L995
It takes some time to understand a complex conditional:
if (status != NodeOperatorStatus.ACTIVE && !(_allActive && (status == NodeOperatorStatus.EJECTED || status == NodeOperatorStatus.JAILED))) continues;
By extracting conditional code to clearly named methods, you are able to reduce a cognitive load for the person who will be maintaining the code later.

Recommendation

To split if to several variables or to a function, remove negations where possible. E.g. bool isActive = status == NodeOperatorStatus.ACTIVE; bool isEjectedOrJailed = status == NodeOperatorStatus.EJECTED || status == NodeOperatorStatus.JAILED; bool getActiveRejectedAndJailed = _allActive; bool hasExtendedStatus = getActiveRejectedAndJailed && isEjectedOrJailed; if (!isActive && !hasExtendedStatus) continue;

9.4.8 Unnecessary Nested ifs Statements

SEVERITY	INFO
STATUS	DISMISSED

Description

NodeOperatorRegistry.sol#L1002

```
if (_delegation) { if (!IValidatorShare(no.validatorShare).delegation()) continue; }
```

Recommendation

To place both checks in one if

9.4.9 Incorrect Term “operator” Instead of “operatorRegistry”

SEVERITY	INFO
STATUS	FIXED

Description

Validator.sol#L26
Validator.sol#L30
address private operator; ... modifier isOperator() {
ValidatorFactory.sol#L21
ValidatorFactory.sol#L26
/// @notice the node operator address. address public operator; ... /// @notice Check if the operator contract is the msg.sender. modifier isOperator() {

Recommendation

To rename to “operatorRegistry” and “isOperatorRegistry”

9.5 Revision 5 Summary

Issue Type	Number of Founded Issues
CRITICAL	3
MAJOR	1
WARNING	0
INFO	9
Total	**13**

10 Revision 6: Findings

10.1 CRITICAL

No critical issues were found

10.2 MAJOR

10.2.1 Unexpected INACTIVE Status for Validator Whos is Jailed and Unstaked Same Time

SEVERITY	MAJOR
STATUS	FIXED

Description

The same issue 9.2.1 might be found in the Revision 5 report.

Recommendation

To see the same issue in the last report.

10.3 WARNING

10.3.1 Impossibility to Withdraw In Case of No Active Validators

SEVERITY	WARNING
STATUS	FIXED

Description

If there are no active validators getMinValidatorBalance will return type(uint256).max because getOperatorInfos(false, false) will return an empty array StMATIC.sol#L707 .
And requestWithdraw function will revert because of overflow in StMATIC.sol#L182

Recommendation

To replace getOperatorInfos(false, false) in getMinValidatorBalance with getOperatorInfos(false, true).
The same as in requestWithdraw StMATIC.sol#L160

10.3.2 “minValidatorBalance” Might Be Greater Than Expected

SEVERITY	WARNING
STATUS	FIXED

Description

minValidatorBalance is calculated for only active validators because of getOperatorInfos(false, false) StMATIC.sol#L707 (See the issue 1 above). While withdrawal in requestWithdraw happens for all staked validators - getOperatorInfos(false, true) StMATIC.sol#L160
It may lead to the locking of some funds of jailed/ejected validators because 10% of the smallest active validator may be bigger than 100% of a jailed/ejected one. Until the DAO unstake or stops the validator, or it calls unjail in case of a jailed validator. Also, it may block withdrawing even so there are enough funds, e.g.:
There is 1 active validator with 100 Matic and 1 jailed/ejected with 10 Matic. A user requests 100 Matic. If both validators were active the function would allow to withdraw up to 99 from the first one and up to 9 from the last, 108 in sum. But in this case, only 90 Matic is allowed to withdraw.

Recommendation

To replace getOperatorInfos(false, false) in getMinValidatorBalance with getOperatorInfos(false, true).
The same as in requestWithdraw StMATIC.sol#L160

10.4 INFO

10.4.1 Unused “WAIT” NodeOperator Status

SEVERITY	INFO
STATUS	FIXED

Description

It is used only in
getState NodeOperatorRegistry.sol#L903

```
} else if (status == NodeOperatorStatus.WAIT) { _totalWaitNodeOperator++; }
```

However, this status is never assigned to an operator.

Recommendation

To remove unused WAIT status.

10.4.2 Duplicating Logic in “unstake” and “stopOperator” Functions

SEVERITY	INFO
STATUS	ACKNOWLEDGED

Description

NodeOperatorRegistry.sol#L414
NodeOperatorRegistry.sol#L220
They do almost the same if you allow stopping an ejected validator.

unstake allows an operator to unstakeClaim and claimFee after, while stopOperator allow migrating

Recommendation

To consider removing unstake(uint256 _operatorId) and allowing to stop an ejected operator in order to save gas and remove duplicated logic.

10.4.3 A Jailed or Ejected Operator will Still Keep minValidatorBalance of Funds

SEVERITY	INFO
STATUS	ACKNOWLEDGED

Description

An evil or an unfortunate validator that is ejected or jailed will have minValidatorBalance of funds locked on its validatorShare until a DAO stops or unstakes it.
Because every validator, even ejected and jailed, will keep minValidatorBalance (StMATIC.sol#L215) until the DAO stops or unstake it.
It will also lead to a little non-optimal distribution because funds are withdrawn in random order, with the same weight for active, ejected and jailed. Even so, active ones generate rewards, and it’s better to keep them.

Recommendation

To consider rewriting the logic, so first, withdraw from ejected validators, then jailed ones and then active ones.

10.4.4 Strict Inequality in withdrawRewards Function

SEVERITY	INFO
STATUS	FIXED

Description

You use > in StMATIC.sol#L383 while in ValidatorShare.sol#L163 >= is used. Because of that even so ValidatorShare allows withdrawing when *rewards* == *rewardThreshold* you do not.

Recommendation

Replace

```
stMaticReward > rewardThreshold
```

with

```
stMaticReward >= rewardThreshold
```

10.4.5 Usage of Comparison for Checking NodeOperatorStatus enum Values

SEVERITY	INFO
STATUS	FIXED

Description

NodeOperatorRegistry.sol#L228
NodeOperatorRegistry.sol#L566
NodeOperatorRegistry.sol#L593

Despite enum values represented by compiler as `uint8` type it is recommended not to use comparison when checking enum values because it might lead to bugs if enum values would be reordered. Moreover it significantly decreases code readability.

Recommendation

To use explicit comparison for needed statuses, eg:

```
status == NodeOperatorStatus.ACTIVE || status == NodeOperatorStatus.INACTIVE
```

instead of

```
status <= NodeOperatorStatus.ACTIVE
```

10.5 Revision 6 Summary

Issue Type	Number of Founded Issues
CRITICAL	0
MAJOR	1
WARNING	2
INFO	5
Total	**8**

11 Final Summary

Smart contracts have been audited and 12 critical and 12 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. The severe attack vectors and potential broken features identified, together with documentation inconsistencies were the reflection of a work-in-progress code base. As stated in each particular issue, all critical, major and warning issues identified have been correctly fixed or acknowledged by the client, so contracts are assumed as secure to use according to our security criteria. Final commit identifier with all fixes: 5be86d9700201397b2fcff13daaaf9549cfe0272. This version is recommended to deploy to testnet for further system testing.

As to further help the project reach a production-ready state, we highly advise additional rounds of security reviews after every change in contracts.

The following table contains total issues found during all audit revisions.

Issue Type	Number of Found Issues
CRITICAL	11
MAJOR	12
WARNING	32
INFO	130
Total	**185**