Launchnodes

LAUNCHNODES LIDO IMPACT STAKING SECURITY AUDIT REPORT

OXORIO

DECEMBER 13, 2024

1 EXECUTIVE SUMMARY

OXORIO

1.1 EXECUTIVE SUMMARY

This document presents the smart contracts security audit conducted by Oxorio for Launchnodes Lido Impact Staking.

Launchnodes provides secure, scalable Ethereum solo staking with pre-synced Geth, Beacon, and Validator nodes in AWS. It simplifies running RPC nodes and offers consultancy services through AWS Marketplace, supporting clients with solo staking and DeFi development.

Lido Impact Staking (LIS) is an innovative platform that allows Ethereum stakers to allocate a portion of their staking rewards to social impact initiatives. Participants can donate part of their returns to approved organizations for a set period while retaining their original capital. LIS operates within the Lido ecosystem, leveraging its staking infrastructure to enable seamless participation in social impact projects.

The audit process involved a comprehensive approach, including manual code review, automated analysis, and extensive testing and simulations of the smart contracts to assess the project's security and functionality. The audit covered a total of 8 smart contracts, encompassing 616 lines of code. The codebase was thoroughly examined, with the audit team collaborating closely with Launchnodes and referencing the provided documentation to address any questions regarding the expected behavior. For an in-depth explanation of used the smart contract security audit methodology, please refer to the Security Assessment Methodology section of this document.

Throughout the audit, a collaborative approach was maintained with `Launchnodes` to address all concerns identified within the audit's scope. Each issue has been either resolved or formally acknowledged by `Lauchnodes`, contributing to the robustness of the project.

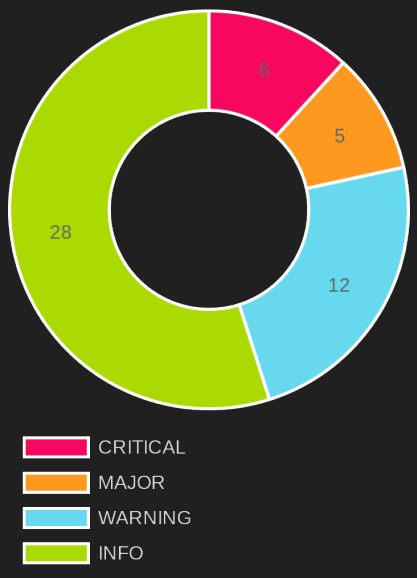
1.2 SUMMARY OF FINDINGS

The table below provides a comprehensive summary of the audit findings, categorizing each by status and severity level. For a detailed description of the severity levels and statuses of findings, see the Findings Classification Reference section.

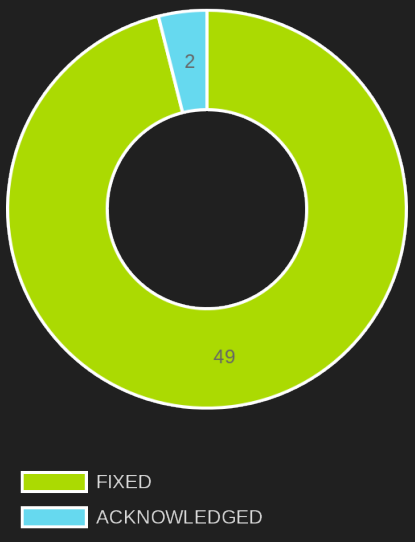
Detailed technical information on the audit findings, along with our recommendations for addressing them, is provided in the Findings Report section for further reference.

All identified issues have been addressed, with `Launchnodes` fixing them or formally acknowledging their status.

Severity	TOTAL	NEW	FIXED	ACKNOWLEDGED	NO ISSUE
CRITICAL	**6**	0	6	0	0
MAJOR	**5**	0	5	0	0
WARNING	**12**	0	11	1	0
INFO	**28**	0	27	1	0
TOTAL	**51**	**0**	**49**	**2**	**0**



Issue distribution by severity



Issue distribution by status

2 AUDIT OVERVIEW

CONTENTS

2.1 DISCLAIMER

At the request of the client, Oxorio consents to the public release of this audit report. The information contained herein is provided "as is" without any representations or warranties of any kind. Oxorio disclaims all liability for any damages arising from or related to the use of this audit report. Oxorio retains copyright over the contents of this report.

This report is based on the scope of materials and documentation provided to Oxorio for the security audit as detailed in the Executive Summary and Audited Files sections. The findings presented in this report may not encompass all potential vulnerabilities. Oxorio delivers this report and its findings on an as-is basis, and any reliance on this report is undertaken at the user's sole risk. It is important to recognize that blockchain technology remains in a developmental stage and is subject to inherent risks and flaws.

This audit does not extend beyond the programming language of smart contracts to include areas such as the compiler layer or other components that may introduce security risks. Consequently, this report should not be interpreted as an endorsement of any project or team, nor does it guarantee the security of the project under review.

THE CONTENT OF THIS REPORT, INCLUDING ITS ACCESS AND/OR USE, AS WELL AS ANY ASSOCIATED SERVICES OR MATERIALS, MUST NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE. Third parties should not rely on this report for making any decisions, including the purchase or sale of any product, service, or asset. Oxorio expressly disclaims any liability related to the report, its contents, and any associated services, including, but not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Oxorio does not warrant, endorse, or take responsibility for any product or service referenced or linked within this report.

For any decisions related to financial, legal, regulatory, or other professional advice, users are strongly encouraged to consult with qualified professionals.

2.2 PROJECT BRIEF

Title	Description
Client	Launchnodes
Project name	Lido Impact Staking
Category	Liquid Staking
Website	https://impactstake.com/
Documentation	https://launchnodes.gitbook.io/lido-impact-staking-lis/
Repository	https://github.com/Launchnodes-Ltd/LIS/
Initial Commit	`1328366ebc49fd40c93a57622678d655bec42ab5`
Reaudited Commit 1	`b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`
Reaudited Commit 2	`e862675353e08d83265337d1944d5d2ca6b6be31`
Reaudited Commit 3	`51f49e43dec8b70f71fb2016c442ff05b198a35b`
Final commit	`51f49e43dec8b70f71fb2016c442ff05b198a35b`
Platform	L1,L2
Languages	Solidity
Lead Auditor	Alexander Mazaletskiy - am@oxor.io
Project Manager	Nataly Demidova - nataly@oxor.io

2.3 PROJECT TIMELINE

The key events and milestones of the project are outlined below.

Date	Event
August 8, 2024	Client engaged Oxorio requesting an audit.
September 18, 2024	The audit team initiated work on the project.
September 24, 2024	Submission of the comprehensive audit report.
October 25, 2024	Client engaged Oxorio requesting an re-audit #1.
October 28, 2024	The audit team initiated work on the re-audit #1.
October 30, 2024	Submission of the comprehensive audit report for re-audit #1.
November 20, 2024	Client engaged Oxorio requesting an re-audit #2.
November 29, 2024	The audit team initiated work on the re-audit #2.
December 2, 2024	Submission of the comprehensive audit report for re-audit #2.
December 6, 2024	Client engaged Oxorio requesting an re-audit #3.
December 10, 2024	The audit team initiated work on the re-audit #3.
December 13, 2024	Submission of the final audit report incorporating client's verified fixes.

2.4 AUDITED FILES

The following table contains a list of the audited files. The scc tool was used to count the number of lines and assess complexity of the files.

	File	Lines	Blanks	Comments	Code	Complexity
1	contracts/interfaces/IAccountOracle.sol	27	2	1	**24**	0
2	contracts/interfaces/IERC20.sol	11	2	5	**4**	0
3	contracts/interfaces/ILido.sol	21	5	1	**15**	0
4	contracts/interfaces/IWithdrawalQueue.sol	36	8	1	**27**	0
5	contracts/libs/ERC6551BytecodeLib.sol	15	1	1	**13**	0
6	contracts/NGOLis.sol	887	137	301	**449**	8
7	contracts/NGOLisFactory.sol	138	13	50	**75**	0
8	contracts/NGOLisProxy.sol	14	4	1	**9**	0
	Total	**1149**	**172**	**361**	**616**	**6**

Lines: The total number of lines in each file. This provides a quick overview of the file size and its contents.

Blanks: The count of blank lines in the file.

Comments: This column shows the number of lines that are comments.

Code: The count of lines that actually contain executable code. This metric is essential for understanding how much of the file is dedicated to operational elements rather than comments or whitespace.

Complexity: This column shows the file complexity per line of code. It is calculated by dividing the file's total complexity (an approximation of cyclomatic complexity that estimates logical depth and decision points like loops and conditional branches) by the number of executable lines of code. A higher value suggests greater complexity per line, indicating areas with concentrated logic.

2.5 PROJECT OVERVIEW

Lido Impact Staking (LIS) integrates with Lido's Ethereum liquid staking infrastructure to create a decentralized platform that funds social impact initiatives using staking returns. LIS leverages Ethereum staking as a sustainable, long-term financing tool, allowing participants to donate a portion of their staking rewards to charitable organizations while retaining their initial capital. The platform is designed to support causes such as poverty alleviation, climate change mitigation, and other social impact efforts, all within a transparent and measurable data-rich context.

LIS operates by connecting to Lido's existing Ethereum liquid staking engine. Stakers can deposit ETH or stETH, and a portion of their staking rewards is diverted to support approved social impact organizations. The core staking and reward distribution is handled by Lido's decentralized network of node operators, ensuring security and scalability.

LIS uses two primary smart contract modules:

- User Module: This contract allows users to stake their ETH or stETH, check their reward balances, and withdraw their initial capital or staking rewards at any time. Users can specify the percentage of their rewards they wish to donate and the duration of their participation in the social impact marketplace.
- Organization Module: Social impact organizations are onboarded through this contract, which enables them to receive donations. Approved organizations can withdraw donations on a rolling basis (every 24 hours) once they are credited. The contract also manages the addition or removal of organizations based on compliance with the platform's standards.

LIS creates a Social Impact Marketplace (SIM) where credible organizations focused on social impact, such as poverty reduction, climate action, and education, can apply to receive donations. Organizations must demonstrate a clear, data-driven impact model and present financial projections that account for the variability in Ethereum staking returns.

Users choose a percentage of their staking rewards to donate for a specified period. These rewards are compounded for the user while a fraction is sent to the designated organizations. LIS ensures that donations are made available to these organizations every 24 hours, providing a steady stream of funding. At the end of the donation period, users retain their original staked capital, underscoring the non-speculative nature of the model.

LIS smart contracts manage the staking, reward distribution, and donation mechanisms, with a focus on minimizing risk and ensuring smooth, transparent operations. The audit process focused on the security of the staking flows, the integrity of the donation mechanism, and the accurate tracking and distribution of funds to social impact organizations.

2.6 CODEBASE QUALITY ASSESSMENT

The Codebase Quality Assessment table offers a comprehensive assessment of various code metrics, as evaluated by our team during the audit, to gauge the overall quality and maturity of the project's codebase. By evaluating factors such as complexity, documentation and testing coverage to best practices, this table highlights areas where the project excels and identifies potential improvement opportunities. Each metric receives an individual rating, offering a clear snapshot of the project's current state, guiding prioritization for refactoring efforts, and providing insights into its maintainability, security, and scalability. For a detailed description of the categories and ratings, see the Codebase Quality Assessment Reference section.

Category	Assessment	Result
Access Control	The project uses the `onlyOwner` modifier for access control. The issues with the `onlyOwner` function have been resolved	**Excellent**
Arithmetic	Potential issues with `underflow` were found in the code. It is also important to pay attention to the precision of calculations and issue `C-04`, `C-05`, `M-02`. These issues have been resolved.	**Good**
Complexity	The code is characterized by minimal redundancy, high readability, and a well-defined architecture.	**Excellent**
Data Validation	The code includes appropriate validations and checks.	**Excellent**
Decentralization	The project does not incorporate a decentralized approach to management, and therefore, the metric is not applicable in this context.	**Not Applicable**
Documentation		**Excellent**
External Dependencies	The project uses `wstETH` to minimize the impact of rebase tokens like `stETH`. However, issue `M-03` highlights a problem with the timeliness of data from the `NGO` pool. This issue with has been resolved.	**Excellent**
Error Handling	The project handles exceptional situations.	**Excellent**
Logging and Monitoring	The project exhibits excellent logging capabilities, recording all important events within the system. This comprehensive logging framework enables the effective use of third-party monitoring services such as `Tenderly` or `Forta`, which facilitate real-time data analysis and enhance the ability to track system performance and security incidents accurately.	**Excellent**

Category	Assessment	Result
Low-Level Calls	All low-level calls within the project are implemented correctly and meet expectations for security and efficiency.	**Excellent**
Testing and Verification	Test coverage can be improved, as the current testing volume was not cover all possible scenarios, leading to some oversights and identified issues. It is recommended to expand the coverage for more accurate detection of potential vulnerabilities and to increase system stability.	**Good**

2.7 FINDINGS BREAKDOWN BY FILE

This table provides an overview of the findings across the audited files, categorized by severity level. It serves as a useful tool for identifying areas that may require attention, helping to prioritize remediation efforts, and provides a clear summary of the audit results.

File	TOTAL	CRITICAL	MAJOR	WARNING	INFO
contracts/NGOLis.sol	**20**	3	1	7	9
contracts/NGOLis.sol	**13**	2	3	2	6
contracts/NGOLis.sol	**10**	0	1	2	7
contracts/NGOLisFactory.sol	**5**	0	0	0	5
contracts/NGOLisFactory.sol	**4**	1	0	1	2
contracts/interfaces/IAccountOracle.sol	**1**	0	0	0	1

2.8 CONCLUSION

A comprehensive audit was conducted on 8 smart contracts, revealing 4 critical and 1 major issues, along with numerous warnings and informational notes. The audit uncovered critical vulnerabilities and security risks, including unauthorized access to key functions, incorrect withdrawal logic, handling of slashing and penalties, and potential underflow errors. Issues related to rounding, gas inefficiencies, and lack of validation were also found.

The proposed changes are focused on strengthening access control mechanisms, ensuring proper handling of slashing and penalties, improving withdrawal logic to prevent fund mismanagement, and addressing rounding and underflow vulnerabilities. Additionally, we recommend code optimization for better gas efficiency and enhanced validation to improve the overall security, functionality, and reliability of the smart contracts. These recommendations are based on adherence to industry best practices, ensuring that these aspects are enhanced to improve the overall security and reliability of the smart contracts. We strongly advise to address the identified issues to mitigate potential risks, improve the quality of the codebase, and ensure the contracts meet the highest security standards.

After the initial audit a comprehensive re-audit #1 was conducted on 8 smart contracts, revealing 2 critical and 3 major issues, along with numerous warnings and informational notes. The re-audit uncovered critical vulnerabilities and security risks, including the ability to restrict operations on the contract, asset value imbalance and potential underflow errors. Issues related to rounding, gas inefficiencies, and lack of validation were also found.

After the re-audit #1, a second re-audit #2 was conducted, revealing additional 2 major issues, alongside 4 warnings and 16 informational notes. The re-audit highlighted issues with minimal withdrawal amounts leading to locked balances, improper validation between different token types, and potential mapping inaccuracies during ownership transfers. Additionally, concerns such as rounding errors, lack of parameter validation, and missing events for critical operations were identified. These findings underline the need for implementing robust validation mechanisms, event logging, and enhanced security practices to mitigate risks effectively.

Upon completion of the re-audit #3, all identified findings have been remediated.

As a result, 51f49e43dec8b70f71fb2016c442ff05b198a35b, operates as intended within the defined scope, based on the information and code provided at the time of evaluation. The robustness of the codebase has been significantly improved, meeting the necessary security and functionality requirements established for this audit.

Moreover, we advise increasing the test coverage of the codebase. Comprehensive testing is essential to uncover edge cases and ensure that the smart contracts perform as expected under various conditions. Enhancing test coverage will not only improve the reliability of the contracts but also contribute to the overall security of the project.

3 FINDINGS REPORT

OXORIO

3.1 CRITICAL

C-01	The `setImplementetion` function is executed without authorization in `NGOLisFactory`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
NGOLisFactory.sol	contract `NGOLisFactory` > function `setImplementetion`	135

Description

In the `setImplementetion` function of the `NGOLisFactory` contract, the implementation of the proxy is set. The proxy itself is created in the `createNGO` function, only by the owner. However, the `setImplementetion` function can be called by anyone.

Thus, an attacker can frontrun the proxy creation transaction by calling the `setImplementetion` function beforehand, setting a different implementation address.

Recommendation

We recommend adding authorization to the `setImplementetion` function to ensure that only a trusted address (e.g., the owner) can change the implementation.

Update

Client's response

Added modifier `onlyOwner` to function `setImplementetion`. From now `onlyOwner` can use function `setImplementetion`.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

C-02	Withdrawal of more ETH from the protocol than returned by Lido in `NGOLis`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `claimWithdrawal`	634

Description

In the function `claimWithdrawal` of contract `NGOLis`, the user withdraws all ETH that was received after the claim in Lido:

```
withdrawalSC.claimWithdrawal(_requestId);

payable(msg.sender).transfer(status.amountOfStETH);
```

The amount of ETH withdrawn is equal to `status.amountOfStETH`, which is the amount of ETH requested for withdrawal in the request. However, according to the documentation, the amount specified in the withdrawal request may not equal the actual claim amount:

> Why is the claimable amount different from my requested amount?
>
> The amount you can claim may differ from your initial request due to slashing and penalties. For these reasons, the total claimable reward amount could be lower than the amount withdrawn.

This leads to the situation where, upon calling the `transfer` function, the user may receive more ETH than what Lido returned if there is excess ETH in the contract. If there is no excess ETH, then the `claimWithdrawal` function will revert with an error.

Additionally, it should be noted that instead of using the `.transfer()` method for sending ETH, it is recommended to use `.call()`.

Recommendation

We recommend considering the use of the function `withdrawalSC.claimWithdrawalsTo` instead of `withdrawalSC.claimWithdrawal`, and directly passing the ETH recipient as one of the arguments. This will simplify the logic and eliminate the need to use the `transfer` function.

Update

Client's response

Removed the transfer function and changed the call function `withdrawalSC.claimWithdrawal` to `withdrawalSC.claimWithdrawalsTo`.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

C-03	Decrease in stETH balance due to slashing or penalties is not accounted for in `NGOLis`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis`	19

Description

In the contract `NGOLis`, a rebasing token `stETH` is used, the balance of which may decrease due to slashing and penalties imposed on Ethereum validators. Since the current balance is obtained directly from `lidoSC.balanceOf(address(this))`, this may lead to errors when working with it. For example:
1) In the `getUserBalance` function, in the event of a decrease in the overall balance `currentBalance`, the calculated user balance may be less than the stake amount. In such a case, the function will return the user's balance equal to the stake:

```
uint256 userTotalShareWithNgoReward = shares[_user][_id].mulDiv(
    currentBalance,
    totalShares
);

if (userTotalShareWithNgoReward < stakedInfo.amount) {
    return stakedInfo.amount;
}
```

It may happen that the overall balance `currentBalance` is less than the total amount of tokens staked by users. Thus, the protocol will incur losses.

At the same time, the loss will not be socialized among the protocol users. That is, the user who first creates a withdrawal request under such conditions will exit the protocol with their stake. Meanwhile, users who create requests last will be unable to withdraw anything.
2) When calculating `_rewardsForToday` in the `handleNGOShareDistribution` function:

```
uint256 _rewardsForToday = currentBalance - stakedBalance - prevRewards;
```

`currentBalance` may become less than `stakedBalance` at a certain point, leading to underflow and failure in the distribution of rewards.
3) In the `withdrawCalculation` function, when `currentBalance` decreases, the calculated value `amountInShares` may exceed the total amount of all shares for the respective stake, leading to underflow and the inability to create a withdrawal request or claim `stETH`:

```
uint256 amountInShares = _amount.mulDiv(
    totalShares + 1,
    currentBalance
);

if (_amount == userBalance) {
    shares[msg.sender][_id] = 0;
} else {
    shares[msg.sender][_id] -= amountInShares;
}
```

Recommendation

We recommend considering the use of a non-rebasing token wstETH instead of the rebasing token `stETH` to avoid complications related to fluctuating balances.

Update

Client's response

Changed `stETH` token for non-rebasing `wstETH` token for usage on smart contract. `WstETH` is used to calculate `NGO` and users' balances. We've enabled an ability to stake and withdraw using `wstETH`, while stakes using `ETH` and `stETH` are being converted to `wstETH`. Withdrawals are also converted to `stETH` or `ETH` if requests are made to withdraw in that form.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

C-04	Rounding during division leads to accumulation of undistributed rewards on the contract in `NGOLis`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `stake`	460
NGOLis.sol	contract `NGOLis` > function `handleNGOShareDistribution`	542
NGOLis.sol	contract `NGOLis` > function `handleNGOShareDistribution`	545

Description

In the mentioned locations, values are calculated using division without the aid of the `mulDiv` library. Thus, during calculations, rounding occurs, which may result in incorrect final values. For example:
1) In the `stake` function, the calculation of `totalShareToday` is performed:

```
stakedBalance += assets;
totalShareToday += (assets * (_ngoPercent)) / PERCENT_DIVIDER;
```

However, in the case of a small `assets` value, the expression `(assets * (_ngoPercent)) / PERCENT_DIVIDER` may equal zero. This will lead to the accumulation of undistributed rewards in `prevRewards`.
2) In the `handleNGOShareDistribution` function, calculations for `shareToNgo` and `_lisFee` are performed. Similarly, with sufficiently small values of `totalShareToday` and `_rewardsForToday`, the results of these calculations may return `0` due to rounding during division, leading to underreported fees and rewards for NGO and the accumulation of undistributed rewards in `prevRewards` on the contract:

```
uint256 shareToNgo = (totalShareToday * _rewardsForToday) /
    stakedBalance;

uint256 _lisFee = (shareToNgo * LIS_FEE) / PERCENT_DIVIDER;
```

Recommendation

We recommend considering the use of the `mulDiv` library to avoid significant rounding errors when dividing small values.

Update

Client's response

Removed all dividers in SC and started using `mulDiv` instead.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

C-05	Contract lock risk on initial stake in `NGOLis`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `stake`	444
NGOLis.sol	contract `NGOLis` > function `stakeStEth`	500
NGOLis.sol	contract `NGOLis` > function `stakeWStEth`	553

Description

In the mentioned locations, a portion of the asset provided by the user, which is allocated to the NGO, is calculated as follows:

```
uint256 _ngoAssets = userAmount.mulDiv(_ngoPercent, PERCENT_DIVIDER);
```

However, there is no check for the minimum stake amount. This allows the user to specify a stake so small that the calculated `_ngoAssets` becomes zero, for instance, with the following variable values:

```
userAmount = 10
_ngoPercent = 100
PERCENT_DIVIDER = 10000
```

If such a small asset is set in the case of the very first stake, where `id == 1`, this leads to setting `totalNGOAssets` and `totalNGOShares` to zero permanently. In this case, no further stakes can be made in the protocol, as for `id > 1`, a division-by-zero error will occur when calculating `_ngoShare`:

```
if (id == 1) {
    _ngoShare = _ngoAssets;
} else {
    _ngoShare = _ngoAssets.mulDiv(totalNGOShares, totalNGOAssets);
}
```

```
totalNGOAssets += _ngoAssets;
totalNGOShares += _ngoShare;
```

Recommendation

We recommend refactoring the code to eliminate the possibility of rounding to zero during division. To achieve this, it is advisable to set a minimum stake size that, even at the lowest `_ngoPercent`, would not result in `_ngoAssets` being calculated as zero.

Update

Client's response

Added modifier `validAmount` and set up a minimum amount possible to stake:

```
uint16 constant MIN_AMOUNT = 1000


modifier validAmount(uint256 amount) {
       if (amount < MIN_AMOUNT) {
           revert InvalidStakeAmount();
       }
       _;
   }
```

Oxorio's response

Fixed in `e862675353e08d83265337d1944d5d2ca6b6be31`.

C-06	Unrestricted withdrawal of small `stETH`/`wstETH` amounts in `NGOLis`
Severity	**CRITICAL**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `withdrawCalculation`	786

Description

In the `withdrawCalculation` function of the `NGOLis` contract, it is possible to pass a very small `_amountWstETH` value when the user's `_totalUserWstETH` stake is very large. In this case, `_ratio` could equal zero due to rounding during division:

```
uint256 _ratio = _amountWstETH.mulDiv(DIVIDER, _totalUserWstETH);
```

For example, with the following variable values:

```
_amountWstETH = 99_999
DIVIDER = 10**18
_totalUserWstETH = 100_000 * 10**18
```

This results in the balance variables (`totalNGOAssets`, `totalNGOShares`, `ngoShares`, etc.) not being updated on the contract, while the requested amount `_amountWstETH` is successfully withdrawn. In this case, we get a mismatch between the assets stored in the contract and the accounting balances in the contract. This means that at some point, a portion of users will not have enough assets to withdraw their stake.

Recommendation

We recommend checking that `_ratio` is not equal to zero after calculation to prevent unrestricted withdrawal of funds from the contract in the `claimWithdrawInStEth` and `claimWithdrawInWStEth` functions.

Update

Client's response

Added a check for ratio == `0`. If a small amount is passed and it occurs that ratio cannot be calculated (due to rounding in Solidity equals to `0`) transaction is being reverted. It resolves issue with cases when `totalNGOassets`, `userAssets` weren't updated, but had to.

```
if (_ratio == 0) {
     revert InvalidWithdrawAmount();
 }
```

Oxorio's response

Fixed in `e862675353e08d83265337d1944d5d2ca6b6be31`.

3.2 MAJOR

M-01	Underflow when attempting to withdraw asset from `prevRewards` in `NGOLis`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `withdrawCalculation`	776

Description

In the function `withdrawCalculation` of contract `NGOLis`, when `_amount <= rewards`, the withdrawal occurs from `prevRewards`:

```
if (_amount > rewards) {
    stakeInfo.amount -= (_amount - rewards);
    stakedBalance -= (_amount - rewards);
    prevRewards = prevRewards > rewards ? prevRewards - rewards : 0;
} else {
    prevRewards -= _amount;
}
```

At this point, `_amount` can be greater than `prevRewards`. For example, immediately after deploying the project, before the first distribution of rewards, `prevRewards` is equal to `0`, but the user may have already staked their assets, accumulated rewards, and wish to withdraw them.

This leads to a situation where, if `_amount` is small relative to the accumulated rewards, `_amount` can simultaneously be greater than `prevRewards`, resulting in an underflow.

Recommendation

We recommend refactoring the accounting of accumulated rewards in `prevRewards` during the withdrawal calculations in the function `withdrawCalculation` to avoid underflow.

Update

Client's response

Changed calculations inside smart contract based on specifications which were provided. Logic of shares for users and NGO was implemented, so the function `handleNGOdistribution` takes into account amount and shares of `wsETH` for `NGO` (prevRewards variable is not used anymore).

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

M-02	Incorrect update of `lastNGOBalance` in `NGOLis`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `stake`	470
NGOLis.sol	contract `NGOLis` > function `stakeStEth`	529
NGOLis.sol	contract `NGOLis` > function `stakeWStEth`	584
NGOLis.sol	contract `NGOLis` > function `withdrawCalculation`	818

Description

In the mentioned locations, there is an incorrect update of the `lastNGOBalance` parameter, resulting in inaccurate calculation of the `NGO` income. This occurs due to the following reasons:

1. `lastNGOBalance` records how many `StETH` tokens correspond to `NGO`, but `StETH` is a rebasing token, and its balance changes over time with each report from Lido.
2. During stake and withdraw operations, the balance of `StETH` is altered, but this change occurs relative to the current state of the `NGO` pool.
3. Updating `lastNGOBalance` resets reward accruals, but it does so based on the current parameters of `lastNGOBalance`, resulting in previous accruals and the state of `lastNGOBalance` being unaccounted for.

Recommendation

We recommend replacing the increment and decrement of `lastNGOBalance` to amount of stake in `stakeStEth`, `stakeWStEth`, and `withdrawCalculation` functions by distributing the accrued rewards and setting `lastNGOBalance` to reflect the current state of the pool like in `handleNGOShareDistribution`.

Update

Client's response

Have made a new function `pendingRewardsCalculation()` that calculates the amount of tokens (in `WStETH` tokens) which should be distributed to NGO with each `handleNGODistribution` handle. It's used in `stake`, `withdrawal` and

`handleNGOdistribution` functions. It helps to store tokens which are meant to be transferred to an NGO and make sure it won't be withdrawn by any user.
With this new logic, we have changed the calculation of user balance in the `getUserBalance` function. Portion of tokens that is meant for the user, but its reward goes to the NGO (`totalNGOAssets`) is being calculated based on `lastNGOBalance`, as `lastNGOBalance` represents an amount without any reward and it is always accurate (taking into account each new stake or withdrawal)

```
function getUserBalance(
    address _user,
    uint256 _id
) public view returns (uint256 userBalance) {
    uint256 _ngoShare = ngoShares[_user][_id];
    uint256 lastNGOBalanceToWsEth = wstETHSC.getWstETHByStETH(
        lastNGOBalance
    );
    uint256 _ngoAssets = _ngoShare.mulDiv(
        lastNGOBalanceToWsEth,
        totalNGOShares
    );

    userBalance = assets[_user][_id] + _ngoAssets;
    userBalance = wstETHSC.getStETHByWstETH(userBalance);
    return userBalance;
}
```

Oxorio's response

Fixed in `e862675353e08d83265337d1944d5d2ca6b6be31`.

M-03	Risk of Zero Calculation for Low `_ratio` and `_ngoAssets` in `NGOLis`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `withdrawCalculation`	789

Description

In the function `withdrawCalculation` of contract `NGOLis`, with sufficiently low values of `_ratio` and `_ngoAssets`, certain multiplications or multiplications of their derivatives may result in a value of `0` due to rounding during division by `DIVIDER`:

```
assets[msg.sender][_id] -= _ratio.mulDiv(
    assets[msg.sender][_id],
    DIVIDER
);
uint256 withdrawnNgoAssets = _ngoAssets.mulDiv(_ratio, DIVIDER);

totalNGOAssets -= withdrawnNgoAssets;

totalNGOShares -= _ngoShare.mulDiv(_ratio, DIVIDER);
ngoShares[msg.sender][_id] -= _ngoShare.mulDiv(_ratio, DIVIDER);
lastNGOBalance -= wstETHSC.getStETHByWstETH(withdrawnNgoAssets);
```

For example, when `_ngoAssets` and `_ratio` are low, the calculation of `withdrawnNgoAssets` may yield `0`. In this case, we get a mismatch between the NGO assets stored in the contract and the accounting balances `totalNGOAssets` in the contract, which leads to an incorrect ratio of `totalNGOShares/totalNGOAssets`.

Recommendation

We recommend setting minimum allowable values for `_ratio` calculated in the `withdrawCalculation` function and for `_ngoAssets` in the staking functions, or checking

the results of multiplications by `_ratio` to prevent zero values due to rounding during division.

Update

Client's response

We've set up a minimum amount to stake, which resolved an issue for small amounts. Minimum amount `1000 wei` with `minimum % = 100` is the smallest possible option for withdrawals. We've also set up a minimum possible amount to withdraw for 100 wei. For this case:
Withdraws 100 wei (minimum allowable amount).
`ratio` = `10^18 * 100/1000` = `10^17`
`withdrawnNGOassets` = `ratio * ngoAssets / divider` = `10^17 * 10 / 10^18` = `1 wei` (which is accurate)

For the case with big amounts minimum amount to withdraw is corrected with new check for ratio != 0:
User staked `100000 * 10^18` wei (`100000` ETH with 1% to NGO)
Withdraws `100000` wei (less amounts will revert transaction as ratio == 0)
ratio = `10^18 * 100000/100000 * 10^18` = 1
`withdrawnNGOassets` = `ratio * ngoAssets` / `divider` = `1 * 1000 * 10^18 / 10^18` = `1000 wei` (which is accurate).

Client's response

Fixed in `e862675353e08d83265337d1944d5d2ca6b6be31`.

M-04	Underflow occurring during validator slashing events in `NGOLis`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `handleNGOShareDistribution`	602

Description

In the `handleNGOShareDistribution` function of the `NGOLis` contract, the current NGO balance in `stETH` is determined using the `getStETHByWstETH` function relative to the accumulated `totalNGOAssets` balance in `wstETH`:

```
uint256 _currentNGOBalance = wstETHSC.getStETHByWstETH(totalNGOAssets);

uint256 _rewardsAssets = wstETHSC.getWstETHByStETH(
    _currentNGOBalance - lastNGOBalance
);
```

However, in the case of validator slashing, there may be less ETH in the Lido contract than there was previously. As a result, the current balance `_currentNGOBalance` may be lower than the previously recorded balance in the variable `lastNGOBalance`.

This would lead to an underflow when attempting to calculate the difference `_currentNGOBalance - lastNGOBalance`.

Recommendation

We recommend implementing a check to ensure that the difference between the current and previous balances does not lead to underflow, and providing a clear error message for the user if it does.

Update

Client's response

New function `pendingRewardsCalculation()` is used for calculations of rewards in `handleNGODistribution`. If `lastNGOBalance` >= `_currentNGOBalance` calculation won't happen. If after the function we've set up a check for `pendingNGOrewards` != `0`. If it equals to `0`, the function would return an error. This also covers the case when `pendingReward` was `0` before `pendingRewardsCalculation()`, as it will affect in the check:
If `pendingNGOrewards` was `0`, and after `pendingRewardsCalculation()` it's still `0`, then `lastNGOBalance` >= `_currentNGOBalance` and contract revert an error.
If we had pending `NGOrewards` was greater than `0`, we should distribute this amount, but `lastNGObalance` will be changed based on change during slashing event (it will be equal to `currentNGOBalance`).

Oxorio's response

Fixed in `e862675353e08d83265337d1944d5d2ca6b6be31`.

M-05	Unwithdrawable balance after withdrawal in `NGOLis`
Severity	**MAJOR**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `requestWithdrawals`	649
NGOLis.sol	contract `NGOLis` > function `claimWithdrawInStEth`	724
NGOLis.sol	contract `NGOLis` > function `claimWithdrawInWStEth`	759

Description

In the function `requestWithdrawals` of contract `NGOLis`, there is a check for the minimum withdrawal amount. However, if the remaining balance on the contract after a withdrawal is less than the minimum, these funds will remain on the contract forever without the possibility of withdrawal.

Recommendation

We recommend refactoring the logic so that the user receives their entire stake if attempting to withdraw an amount that would leave an unwithdrawable balance. This will ensure that no "dust" remains after a withdrawal.

Update

Client's response

In function `withdrawCalculation()` we have added a check for the amount which is being withdrawn.
For these purposes, we have created a new constant `WITHDRAW_GAP` (equals to 100 wei). It represents the min amount that can be stored for each user. If after withdrawal the amount less than `WITHDRAW_GAP` remains for user, we push remainder to be withdrawn (so user's balance is fully withdrawn).

```
if (userBalanceInStEth - _amount < WITHDRAW_GAP) {
    _amountWstETH = wstETHSC.getWstETHByStETH(userBalanceInStEth);
} else {
```

```
        _amountWstETH = wstETHSC.getWstETHByStETH(_amount);
    }
```

Oxorio's response

1) Incorrect calculation

in a situation where `user Balance In StEth` is less than `WITHDRAW_GAP`, the value should be
`` `total User W stETH `` , because we take the entire balance of `wasteth`, and in this case the `ratio` will be equal to 1.

But in the current code, this may not happen because in the current code it is considered like this

```
    uint256 userBalanceInStEth = getUserBalance(msg.sender, _id);
    _amountWstETH = wstETHSC.getWstETHByStETH(userBalanceInStEth);
```

at the same time getUserBalance counts like this:

```
function getUserBalance(
        address _user,
        uint256 _id
    ) public view returns (uint256 userBalance) {
        uint256 _ngoShare = ngoShares[_user][_id];
        uint256 lastNGOBalanceToWsEth = wstETHSC.getWstETHByStETH(
            lastNGOBalance
        );
        uint256 _ngoAssets = _ngoShare.mulDiv(
            lastNGOBalanceToWsEth,
            totalNGOShares
        );


        userBalance = assets[_user][_id] + _ngoAssets;


        userBalance = wstETHSC.getStETHByWstETH(userBalance);


        return userBalance;
}
```

where `ngoAssets` is already calculated based on `lastNGOBalanceToWsEth` and therefore `_amountWstETH` will not be equal to `totalUserWstETH`.

2) Redunant calls

The current code has a huge number of unnecessary calls, function calls are becoming expensive, every time a call is made to the `getUserBalance`, `getWstETHByStETH` and `getWstETHByStETH`, there is an external call to the `wstETH` contract.

it's in the function:

```
function withdrawCalculation(
    uint256 _amount,
    uint256 _id
) private returns (uint256 _amountWstETH) {
    uint256 _ngoShare = _ngoShares[msg.sender][_id];
    pendingRewardsCalculation();
    uint256 _ngoAssets = _ngoShare.mulDiv(_totalNGOAssets, _totalNGOShares);
    uint256 _totalUserWstETH = _assets[msg.sender][_id] + _ngoAssets;
    // 2 external call
    uint256 userBalanceInStEth = getUserBalance(msg.sender, _id);


    // 1 external call
    if (userBalanceInStEth - _amount < WITHDRAW_GAP) {
        _amountWstETH = wstETHSC.getWstETHByStETH(userBalanceInStEth);
    } else {
        _amountWstETH = wstETHSC.getWstETHByStETH(_amount);
    }

    uint256 _ratio = _amountWstETH.mulDiv(DIVIDER, _totalUserWstETH);
    if (_ratio == 0) {
        revert InvalidWithdrawAmount();
    }

    _assets[msg.sender][_id] -= _ratio.mulDiv(
        _assets[msg.sender][_id],
        DIVIDER
    );

    uint256 withdrawnNgoAssets = _ngoAssets.mulDiv(_ratio, DIVIDER);

    _totalNGOAssets -= withdrawnNgoAssets;
```

```
        _totalNGOShares -= _ngoShare.mulDiv(_ratio, DIVIDER);
        _ngoShares[msg.sender][_id] -= _ngoShare.mulDiv(_ratio, DIVIDER);

        // 1 external call
        _lastNGOBalance = wstETHSC.getStETHByWstETH(_totalNGOAssets);

        return (_amountWstETH);
    }
```

whereby the value `userBalanceInStEth` can be passed as a parameter, since the `getUserBalance(msg.sender, _id)` occurs in all three calls to `requestWithdrawals`, `claimWithdrawInWStEth`, `claimWithdrawInStEth`.

Client's response

In function `WithdrawCalculation()` we use user balance from function `getUserBalance()` that equals to user balance calculated based on `lastNGOBalance`.

Conversion of `_totalUserWstETH` cannot be equal to `_userBalanceInStEth` as we use LIDO protocol which have 1-2 wei issue docs.lido.fi/guides/lido-tokens-integration-guide/#1-2-wei-corner-case.

It can affect calculations inside smart contract and disable withdrawals for users.

We understand that dust (1-4 wei) can remain on users balances after full withdrawal, as we take into account LIDO's 1-2 wei issue.

For the function `WithdrawCalculation()` we have added a new parameter `_userBalanceInStEth`. It replicates `getUserBalance()` function usage (is called before `withdrawCalculation()`).

Oxorio's response

Fixed in `51f49e43dec8b70f71fb2016c442ff05b198a35b`.

3.3 WARNING

W-01	Possible to pass a zero `_amount` in `NGOLis`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `claimWithdrawInStEth`	653

Description

In the function `claimWithdrawInStEth` of contract `NGOLis`, it is possible to pass `_amount == 0`. This results in the ability to spam the frontend and monitoring systems with `WithdrawInStEthClaimed` events, paying only for the transaction gas.

Recommendation

We recommend adding a check for the `_amount` value to prevent zero values.

Update

Client's response

Added a check for the `_amount` which is passed to `claimWithdrawInStEth` function and added the revert “ZeroAmount” when `amount == 0` is passed.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

W-02	Inconsistency in the calculation of `userTotalShareWithNgoReward` and `rewardToNgo` in `NGOLis`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `getUserBalance`	723

Description

In the function `getUserBalance` of contract `NGOLis`, when calculating `rewardToNgo`, the expression `(shares[_user][_id] * currentBalance) / totalShares` is computed. However, this expression has already been computed for the variable `userTotalShareWithNgoReward`:

```
uint256 userTotalShareWithNgoReward = shares[_user][_id].mulDiv(
    currentBalance,
    totalShares
);

// ...

rewardToNgo =
    ((((shares[_user][_id] * currentBalance) / totalShares) -
        stakedInfo.amount) * stakedInfo.percent) /
    PERCENT_DIVIDER;

userTotal = userTotalShareWithNgoReward - rewardToNgo;
```

Moreover, the variable `userTotalShareWithNgoReward` is calculated using the `mulDiv` library, while `rewardToNgo` is not.

This can lead to a situation where, due to rounding in the calculation of `rewardToNgo`, the final value of `userTotal` is greater than it should be with a more accurate calculation using the `mulDiv` library.

Recommendation

We recommend using the already calculated value of `userTotalShareWithNgoReward` when calculating the variable `rewardToNgo` instead of the expression `(shares[_user][_id] * currentBalance) / totalShares`. This will make the code cleaner, more gas-efficient, and eliminate inaccuracies in calculating `userTotal`.

Update

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

W-03	User does not receive rewards for staking when `shares == 0` in `NGOLis`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `getUserBalance`	706

Description

In the function `getUserBalance` of contract `NGOLis`, if `shares == 0`, the function returns the user's balance equal to the amount of their original stake. This means that a user with `shares == 0` can only withdraw their stake back, while all accumulated rewards for that stake remain in the contract and cannot be withdrawn by anyone.

Recommendation

We recommend considering the addition of accounting for accumulated rewards in cases where the user has `shares == 0` and `stakedInfo.amount != 0`, to avoid a situation where tokens may get stuck in the contract with no possibility of withdrawal.

Update

Client's response

New formulas were implemented for the function `getUserBalance` based on specification. Unnecessary calculations were removed.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

W-04	Possible to receive a zero amount of shares when converting non-zero assets in `NGOLis`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `convertAssetsToShares`	425

Description

In the function `convertAssetsToShares` of contract `NGOLis`, the number of shares is calculated based on the current values of `totalShares` and `totalAssets`:

```
return (assets * totalShares) / totalAssets;
```

However, `totalAssets` is set immediately before the call to the function `convertAssetsToShares` and is equal to the current balance of the contract in stETH.

Thus, if `totalAssets` is greater than `(assets * totalShares)`, the conversion function will return `0` shares.

This is possible, for example, if an inattentive user accidentally transfers a significant amount of their stETH to the address of the contract `NGOLis`. In this case, protocol users invoking staking functions such as `stake/stakeStEth` will end up receiving `0` shares.

Recommendation

We recommend considering refactoring the logic so that the accounting of the `stETH` balance occurs on the contract, rather than obtaining it at the moment via the external call `balanceOf`. This could prevent users from attempting to manipulate the contract balance.

Update

Client's response

Function `convertAssetsToShares` was removed new functionality with `wstETH` and shares for `NGO` is used.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

W-05	Proxy creation uses `ERC1967Proxy` instead of `NGOLisProxy` in `NGOLisFactory`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NGOLisFactory.sol	contract `NGOLisFactory` > function `createNGO`	95

Description

In the function `createNGO` of contract `NGOLisFactory`, a proxy is created based on the contract `ERC1967Proxy`, rather than the contract `NGOLisProxy`.

Recommendation

We recommend changing the contract for proxy creation from `ERC1967Proxy` to `NGOLisProxy`, or removing the `NGOLisProxy` contract if it is not used.

Update

Client's response

Contract `NGOLisProxy` was removed. `ERC1967Proxy` is used.

Oxorio's response

Not fixed, the contract `NGOLisProxy` exists in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`

Client's response

Was deleted with the commit e862675353e08d83265337d1944d5d2ca6b6be31.

Oxorio's response

Fixed in e862675353e08d83265337d1944d5d2ca6b6be31.

W-06	Function `__ReentrancyGuard_init` is not called during initialization in `NGOLis`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `initialize`	393

Description

In the function `initialize` of contract `NGOLis`, the function `__ReentrancyGuard_init` is not called to initialize the inherited contract `ReentrancyGuardUpgradeable`.

Recommendation

We recommend adding a call to the function `__ReentrancyGuard_init` in the function `initialize`.

Update

Client's response

Added `__ReentrancyGuard_init` call to the function `initialize`.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

W-07	Insufficient validation of the size of `_amount` for withdrawal requests in `NGOLis`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `requestWithdrawals`	587

Description

In the function `requestWithdrawals` of contract `NGOLis`, a request for withdrawal is made for the size of `_amount`. However, in the called external contract `withdrawalSC`, there is a check on the size of the passed `_amount`:

```
if (_amountOfStETH < MIN_STETH_WITHDRAWAL_AMOUNT) {
    revert RequestAmountTooSmall(_amountOfStETH);
}
if (_amountOfStETH > MAX_STETH_WITHDRAWAL_AMOUNT) {
    revert RequestAmountTooLarge(_amountOfStETH);
}
```

This leads to the situation where the user will receive an error if they pass an out-of-bounds size for `_amount`, but only after a series of calculations have been performed.

Recommendation

We recommend adding checks for the minimum and maximum allowable size of `_amount` for withdrawal requests at the beginning of the function. It is also advisable to consider adding additional logic that would break the request into several separate requests if the size of `_amount` exceeds the established limit.

Update

Client's response

Added checks for `MIN` and `MAX` withdrawal amount in the function `requestWithdrawals`.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

W-08	Users cannot withdraw stuck funds from the contract in `NGOLis`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `setUserBan`	857

Description

In the function `setUserBan` of contract `NGOLis`, it is possible to ban a user. In such a case, their funds become frozen in the contract. If these funds were obtained by the user through illegal means, the admin cannot withdraw them.

The same applies to the funds of inattentive users who accidentally transferred their ETH or any other tokens to the contract.

Recommendation

We recommend implementing a function in the contract that would allow the admin to withdraw stuck tokens or excess ETH that was not staked in the manner prescribed in the contract.

Update

Client's response

We appreciate the suggestion to implement an admin function to withdraw stuck tokens or excess `ETH`. However, based on our protocol's philosophy, user funds should always remain accessible solely by the user, even in cases where they may no longer be able to participate in staking due to a ban.

To maintain user control over their staked funds, we implemented a solution that allows banned users to withdraw their staked funds but prevents them from staking additional funds. This ensures that banned users retain ownership and withdrawal rights over their funds, preserving security and user autonomy while still meeting the restriction objectives. Launchnodes can never have custody or access to `LIS` users staked assets.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

W-09	Incorrect value in the `WithdrawClaimed` event in `NGOLis`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `claimWithdrawal`	705

Description

In the `claimWithdrawal` function of the `NGOLis` contract, the `WithdrawClaimed` event uses an incorrect value for amount. The event description states:

```
/**

 * @dev Emitted when a user claims a withdrawal.
 * @param _claimer The address of the user claiming withdrawal.
 * @param _ngo The address of the NGO contract.
 * @param _amount The amount of ETH claimed.
 * @param _requestId The ID of the withdrawal request.
 * @param _timestamp The block timestamp when withdraw was claimed.
 * @param _blockNumber The block number when withdraw was claimed.
 */
```

However, instead of the amount in `ETH`, the parameter `status.amountOfStETH` is passed, which reflects the amount requested for withdrawal in `stETH`, but not the amount actually received by the user. This could lead to incorrect interpretation of the parameters by those working with the protocol.

Recommendation

We recommend either updating the documentation or correcting the value to the accurate `ETH` amount that the user receives.

Update

Oxorio's response

Changed documentation (displaying `stETH` data from LIDO instead of `ETH`).

Client's response

Fixed in `e862675353e08d83265337d1944d5d2ca6b6be31`.

W-10	`StakeInfo` is not updated during withdraw operations in `NGOLis`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `withdrawCalculation`	823

Description

In the function `withdrawCalculation` of the `NGOLis` contract, the `StakeInfo` structure within the mapping `_userToStakeInfo` is not being updated. According to the documentation:

```
/**

 * @dev Struct representing stake information for a user.
 */
struct StakeInfo {
    uint16 percent;
    uint amount;
    uint startDate;
}
```

However, the `amount` parameter is not updated during `withdrawCalculation`. This results in the `getUserStakeInfo` function reflecting incorrect information about the user's stake.

Recommendation

We recommend updating the `StakeInfo` in the `_userToStakeInfo` mapping to ensure accurate reflection of a user's stake information.

Update

Client's response

`StakeInfo` displays initial data for stake. Updated documentation.

Oxorio's response

Fixed in `e862675353e08d83265337d1944d5d2ca6b6be31`.

W-11	Equality of minimum values for different tokens `wstEth` and `stEth` in `NGOLis`
Severity	**WARNING**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `stake`	516
NGOLis.sol	contract `NGOLis` > function `stakeStEth`	555
NGOLis.sol	contract `NGOLis` > function `stakeWStEth`	589
NGOLis.sol	contract `NGOLis` > function `claimWithdrawInStEth`	724
NGOLis.sol	contract `NGOLis` > function `claimWithdrawInWStEth`	759

Description

In the mentioned locations, the modifiers `validAmount` and `validWithdrawalAmount` receive the `amount` parameter, which is then checked against the constants `MIN_AMOUNT` and `MIN_WITHDRAWAL_AMOUNT`, respectively. However, the functions accept amounts of tokens of different types.

This results in `stEth` and `wstEth` tokens being checked against the same minimum values (`MIN_AMOUNT` and `MIN_WITHDRAWAL_AMOUNT`), even though the tokens have different dimensions.

Recommendation

We recommend adding separate minimum values for each token type for validation.

Update

Client's response

Constant variable `MIN_AMOUNT` represents minimum amount in wei (equals to 1000). This variable is used in modifier validAmount() for stake, stakeStEth to check if amount passed to stake is not less than minimum amount (constant variable in wei). It's used for both native `ETH` and `stETH` (as `stETH` represents `ETH` amount and they're meant to be equaled)

For function `stakeWStEth` new `revert()` to check if the amount in `WStEth` passed is valid. In the revert we check if the amount in WStEth converted to StEth is not less than `MIN_AMOUNT` in wei (it's made to check conversion amount, as conversion of WStEth to StEth is not static, and can be increased/decreased with time).

Constant variable `MIN_WITHDRAWABLE_AMOUNT` represents the minimum amount in wei that could be withdrawn in wei (equals to 100). This variable is used on modifier `validWithdrawalAmount` in function `claimWithdrawInStEth()`.

In function `claimWithdrawInWStEth()` we added a revert in check if passed amount (converted to `StEth`) is not less than `MIN_WITHDRAWABLE_AMOUNT`.

With new changes, for each token type we have set up different minimum amounts (using reverts or modifiers).

Oxorio's response

Fixed in `51f49e43dec8b70f71fb2016c442ff05b198a35b`

W-12	Zero `ratio` after rounding during division in `NGOLis`
Severity	**WARNING**
Status	• ACKNOWLEDGED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `withdrawCalculation`	810

Description

In the function `withdrawCalculation` of contract `NGOLis`, it is possible for the `ratio` to equal zero. For example, if `_amountWstETH = 999` and `_totalUserWstETH` exceeds `1000 * 10^18`.

In this case, some amount of tokens cannot be withdrawn from the contract:

```
if (_ratio == 0) {
    revert InvalidWithdrawAmount();
}
```

Recommendation

We recommend revising the withdrawal calculation logic to prevent a scenario where `ratio == 0`.

Update

Client's response

Added this edge case scenario in our documentation here launchnodes.gitbook.io/lido-impact-staking-lis/contracts/ngolis#security-features.

3.4 INFO

I-01	Delay between reward distributions changes after contract initialization in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `handleNGOShareDistribution`	564

Description

In the function `handleNGOShareDistribution` of the contract `NGOLis`, the accumulated rewards are distributed only if the current time is greater than or equal to `lastCountRewardsTimestamp`:

```
if (block.timestamp < lastCountRewardsTimestamp)
    revert TimeNotPassed(block.timestamp, lastCountRewardsTimestamp, 0);

// ...

lastCountRewardsTimestamp += 1 hours;
```

After the distribution of rewards, `lastCountRewardsTimestamp` is increased by one hour. Thus, rewards can be distributed no more frequently than once an hour.

However, during the initialization of the contract, according to the function `getRoundDate`, the variable `lastCountRewardsTimestamp` is set to the beginning of the current day, rather than the hour:

```
function getRoundDate(uint _timestamp) private pure returns (uint) {
    return (_timestamp / 1 days) * 1 days;
}
```

As a result, there is a possibility that immediately after the contract initialization, the reward distribution may be called multiple times in a row. A similar situation may arise if the function `handleNGOShareDistribution` has not been called for more than two hours.

This leads to a distortion in the recording of historical data, as the time `lastCountRewardsTimestamp` will not correspond to the rewards earned in the hour:

```
_historyRewards[lastCountRewardsTimestamp] = _rewardsForToday;
_historyStakedBalance[lastCountRewardsTimestamp] = stakedBalance;
_historyBalance[lastCountRewardsTimestamp] = currentBalance;
```

It is also worth noting that variable names such as `totalShareToday` and `_rewardsForToday` may be misleading, as they imply a daily distribution.

Recommendation

We recommend eliminating the inconsistency between setting `lastCountRewardsTimestamp` during contract initialization and its subsequent use, as well as considering renaming the variables `totalShareToday` and `_rewardsForToday` to avoid confusion.

Update

Client's response

Removed a check for last time reward because now it is not necessary with new `wstETH` logic. Variables `totalShareToday` and `_rewardsForToday` were removed.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`

I-02	Setting `msg.sender` instead of `owner` in the mapping `ownerToNgo` in `NGOLisFactory`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLisFactory.sol	contract `NGOLisFactory` > function `createNGO`	107

Description

In the function `createNGO` of contract `NGOLisFactory`, a proxy contract for the NGO is created and the address `owner` is passed as the owner of this contract. However, in the mapping `ownerToNgo` for the created NGO contract, the address `msg.sender` is set as the owner instead of `owner`.

It should also be noted that the mapping `ownerToNgo` cannot be changed later. At the same time, the NGO owner's address can be changed by calling `transferOwnership`.

Recommendation

We recommend changing the owner set for the NGO in the mapping `ownerToNgo` from `msg.sender` to `owner`, as well as considering adding logic for changing the owners of NGOs in the mapping `ownerToNgo`.

Update

Client's response

Removed mapping `ownerToNgo` due non-unusage of unread method and inside of smart contract.

Oxorio's response

Fixed in `51f49e43dec8b70f71fb2016c442ff05b198a35b`

I-03	Inconsistency in error notification methods in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `claimWithdrawal`	630

Description

In the function `claimWithdrawal` of contract `NGOLis`, `require` is used to check a condition. However, throughout the rest of the contract code, the `revert` call is used in case of errors.

Recommendation

We recommend replacing `require` with a `revert` call to maintain a consistent style of error notifications.

Update

Client's response

Replaced all `require` with a revert calls to maintain a consistent style of error notifications.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

I-04	No `_disableInitializers` call in the constructor in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `initialize`	19

Description

In the constructor of contract `NGOLis`, there is no call to the `_disableInitializers` method.

The absence of the `_disableInitializers` method could potentially allow an attacker to gain administrative rights in the implementation contract by calling the `initialize` function. After obtaining such rights, they could use them to perform, for example, phishing attacks.

Recommendation

We recommend invoking the `_disableInitializers` in the constructor of `NGOLis`.

Update

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

I-05	Redundant storage of the `totalAssets` value in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `stake`	436
NGOLis.sol	contract `NGOLis` > function `stakeStEth`	486

Description

In the mentioned locations, the storage variable `totalAssets` is set. It is used only within a single function and is private. Thus, there is an expensive write and read operation from storage instead of storing this value in memory.

Recommendation

We recommend removing the `totalAssets` variable from storage and storing its value in memory when working with the `stake` and `stakeStEth` functions.

Update

Client's response

Removed unnecessary variable `totalAssets` as new logic was implemented.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

I-06	Suboptimal handling of storage variables in `NGOLis`
Severity	**INFO**
Status	• FIXED

Description

In the contract `NGOLis`, in several cases, the same variables from storage are read and rewritten multiple times during the function call. For example, the variable `lastCountRewardsTimestamp` in the function `handleNGOShareDistribution`.

This leads to suboptimal gas consumption when working with such functions.

Recommendation

We recommend loading frequently used storage variables into memory during function calls, as working with memory is more efficient for the end user.

Update

Client's response

`lastCountRewards` was removed from the smart contract as it's not needed for logic of the project. We have removed all such variables (there were 3 of them) For now, based on specification, we made variables optimized.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

I-07	No parameter validation in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `initialize`	393

Description

In the function `initialize` of contract `NGOLis`, the input parameters are not validated. For example, the passed addresses can be equal to `0`, which would lead to setting incorrect values for state variables.

Recommendation

We recommend adding validation for the input parameters.

Update

Client's response

Added validation for zero address in the function `initialize`.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

I-08	Interface not used in `IAccountOracle.sol`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
IAccountOracle.sol	interface `IAccountOracle`	4

Description

In the file `IAccountOracle.sol`, the interface `IAccountOracle` is defined, which is not used in the project.

Recommendation

We recommend removing the interface `IAccountOracle` to maintain codebase cleanliness.

Update

Client's response

Unnecessary interface `IAccountOracle` was removed.

Oxorio's response

Fixed in `e862675353e08d83265337d1944d5d2ca6b6be31`.

I-09	Unused variable `_prevRewards` in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis`	278

Description

In the `NGOLis` contract, the variable `_prevRewards` is defined but not used. At the same time, there is a similar variable `prevRewards` that is actively used.

Recommendation

We recommend removing the `_prevRewards` variable to maintain code clarity and eliminate ambiguity.

Update

Client's response

Unnecessary variable `_prevRewards` was removed.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

I-10	Redundant check for unsigned value being negative in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `handleNGOShareDistribution`	540

Description

In the `handleNGOShareDistribution` function of the `NGOLis` contract, there is a check to ensure that `_rewardsForToday` is not less than `0`:

```
if (_rewardsForToday <= 0) revert RewardError();
```

However, the `_rewardsForToday` variable is of type `uint256`, an unsigned type that cannot be less than `0`.

Recommendation

We recommend removing the check for `_rewardsForToday < 0` and keeping only the check for equality to zero.

Update

Client's response

Removed the check `_rewardsForToday < 0`.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

I-11	Balance calculation before checking for stake existence in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `withdrawCalculation`	749

Description

In the `withdrawCalculation` function of the `NGOLis` contract, the check for a non-zero `amount` of the stake occurs after calling the `getCurrentBalanceFromLido` and `getUserBalance` functions:

```
StakeInfo storage stakeInfo = _userToStakeInfo[msg.sender][_id];

uint currentBalance = getCurrentBalanceFromLido();

uint256 userBalance = getUserBalance(msg.sender, _id);

if (stakeInfo.amount == 0) {
    revert NotStaked();
}
```

This results in unnecessary computations being performed before the function reverts, in case `stakeInfo.amount == 0`.

Recommendation

We recommend moving the check for a zero `stakeInfo.amount` to the beginning of the function.

Update

Client's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`

Client's response

Removed the check for zero amount as other functions have validations before triggering withdrawCalculation:

```
    if (_amount == 0) {
            revert ZeroAmount();
        }
or
  if (_amount < withdrawalSC.MIN_STETH_WITHDRAWAL_AMOUNT()) {
            revert RequestAmountTooSmall(_amount);
        }
```

I-12	Insufficient validation in the case of the very first stake in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `stake`	455
NGOLis.sol	contract `NGOLis` > function `stakeStEth`	504

Description

In the mentioned locations, in the case of the very first stake, that is when `totalShares` equals zero, the input value `assets` is not validated. It is converted to `share` on a one-to-one basis, which cannot be less than `1000`. Otherwise, an underflow will occur:

```
uint256 share = convertAssetsToShares(assets);

if (totalShares == 0) {
    // ...
    share -= 1000;
}
```

Thus, under these conditions, if a user passes `assets < 1000`, they will receive an unreadable error.

Recommendation

We recommend explicitly validating the converted value of `share` and returning a clear error message to the user if, when `totalShares == 0`, the value of `share` is less than `1000`.

Update

Client's response

The validation was removed, as new `wsETH` logic was implemented based on specifications.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

I-13	Event name in the specification is misleading in `NGOLisFactory`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLisFactory.sol	contract `NGOLisFactory`	123

Description

In the `withdrawFeeStEth` function of the contract `NGOLisFactory`, the event name in the specification does not match the actual emitted event within the function:

```
 * @notice Emit [NGOCreated](#ngocreated) event
 */
function withdrawFeeStEth() public onlyOwner {
    // ...
    emit LisFeeClaimed(_balanceForWithdraw);
}
```

Recommendation

We recommend changing the event name in the specification from `NGOCreated` to `LisFeeClaimed`.

Update

Client's response

Event name `NGOCreated` was changed to correct `LisFeeClaimed`.

Oxorio's response

Fixed in `b2d4294584ab74093b5b3e9f7f00ae9ed9fcd216`.

I-14	Similar code in identical functions in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `stake`	444
NGOLis.sol	contract `NGOLis` > function `stakeStEth`	500
NGOLis.sol	contract `NGOLis` > function `stakeWStEth`	553

Description

In the mentioned locations, most of the code in the three specified functions is identical. The code starting with the calculation of `_ngoAssets` could be extracted into a separate function.

Recommendation

We recommend considering the extraction of the common code in the `stake`, `stakeStEth`, and `stakeWStEth` functions into a separate private function to optimize and simplify codebase maintenance.

Update

Client's response

Created new private function `assetsCalculation(uint256 _amount, uint16 _percent)` private for calculations in other functions.

Oxorio's response

Fixed in `e862675353e08d83265337d1944d5d2ca6b6be31`.

I-15	Transfer of zero `_fee` is possible in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `handleNGOShareDistribution`	612

Description

In the function `handleNGOShareDistribution` of contract `NGOLis`, with sufficiently low values of `_rewardsAssets`, there may be a situation where due to rounding during division, the value of `_fee` will be zero:

```
uint256 _fee = _rewardsAssets.mulDiv(LIS_FEE, PERCENT_DIVIDER);

wstETHSC.transfer(_lis, _fee);
```

In this case, it is meaningless to perform a transfer of zero `_fee`. Moreover, this will lead to the emission of events regarding the conducted transfer.

Recommendation

We recommend not to invoke the transfer when `_fee` is zero, to avoid unnecessary external calls and to prevent the creation of unnecessary events for transferring `0` funds.

Update

Client's response

Added a check for fee amount and won't invoke transfer fee if it equals to 0:

```
if (_fee != 0) {
    wstETHSC.transfer(_lis, _fee);
}
```

Oxorio's response

Fixed in `e862675353e08d83265337d1944d5d2ca6b6be31`.

I-16	No setter parameter validation in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `setOracle`	843
NGOLis.sol	contract `NGOLis` > function `setRewardsOwner`	851

Description

In the mentioned locations, the input parameters are not validated. For example, the passed addresses can be equal to `0`, which would lead to setting incorrect values for state variables.

Recommendation

We recommend adding validation for the input parameters, as is done for the `initialize` function, to maintain consistency in checks.

Update

Client's response

Added validation to zero address in function `setOracle`, `setRewardsOwner`.

Oxorio's response

Fixed in `e862675353e08d83265337d1944d5d2ca6b6be31`.

I-17	Simultaneous use of `uint` and `uint256` types
Severity	**INFO**
Status	• FIXED

Description

In many contracts across the project, both `uint` and `uint256` are used simultaneously for defining variable types. For example:

```
function claimWithdrawInWStEth(uint256 _amount, uint _id) public {
```

Recommendation

We recommend using a single type for variable definitions to maintain a consistent style across the project. This practice improves code readability and maintainability by reducing confusion and potential type conversion issues.

Update

Client's response

Changed `uint` to `uint256` types for the project.

Oxorio's response

Fixed in `e862675353e08d83265337d1944d5d2ca6b6be31`.

I-18	Redundant increment operation in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `stake`	449
NGOLis.sol	contract `NGOLis` > function `stake`	466
NGOLis.sol	contract `NGOLis` > function `stakeStEth`	508
NGOLis.sol	contract `NGOLis` > function `stakeStEth`	525
NGOLis.sol	contract `NGOLis` > function `stakeWStEth`	561
NGOLis.sol	contract `NGOLis` > function `stakeWStEth`	579

Description

In the mentioned locations, the increment operator (`+=`) is used for the value. However, each new operation is recorded under a separate `id`, and the value of the previous stake can only change upon withdrawal.

Recommendation

We recommend replacing the `+=` operator with the `=` operator.

Update

Client's response

Replaced the `+=` operator with the `=` operator.

Oxorio's response

Fixed in `e862675353e08d83265337d1944d5d2ca6b6be31`.

I-19	Suboptimal computation of user balance in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `getUserBalance`	849
NGOLis.sol	contract `NGOLis` > function `withdrawCalculation`	795

Description

In the mentioned locations, the user's balance in `wstETH` tokens is calculated. However, the same balance is already reflected in the `_userToStakeInfo` mapping and the `StakeInfo` structure.

Recommendation

We recommend using the `_userToStakeInfo` mapping and the amount value from the `StakeInfo` structure instead of recalculating userBalance.

Update

Client's response

`UserStakeInfo` is used as info about initial stake. This data is not being updated, that's why we cannot use this method for other calculations.

Oxorio's response

Fixed in `e862675353e08d83265337d1944d5d2ca6b6be31`.

I-20	Missing functionality for mistaken tokens and `ETH` withdrawal in `NGOLis`
Severity	**INFO**
Status	• ACKNOWLEDGED

Location

File	Location	Line
NGOLis.sol	-	-

Description

The NGO contract lacks functionality for withdrawing tokens and `ETH` that were transferred by mistake.

Recommendation

We recommend removing the `receive() external payable {}` function and adding a separate function that allows for the withdrawal of tokens mistakenly received by the contract, without affecting the staked balance of wstETH.

Update

Client's response

We are interpreting this as a requirement to the remove functions `receive() external payable {} function`. In order to do this we'll need to rewrite other methods (such as `Stake`), as the function is used there .

We think u are suggesting we create functionality for some kind of "admins" , as we cannot store data inside smart contract about transfers, as functions and formulas of smart contract are not being triggered.

We do not want to do this and we do not want to create a separate function for the smart contract to allow funds to be returned in the event that someone sends funds to the smart contract accidentally instead of staking on LIS. The reasons are

This functionality becomes an attack vector for each social impact project.
Anyone sending funds to a smart contract is warned by their wallet that they are sending their funds to a smart contract not a wallet and may lose their funds.
We will re-iterate this warning in our documentation and ask impact stakers to contact us if

they need support depositing to `LIS`.
We think this is more about application functionality than security. We would welcome your feedback on our analysis of this point.

I-21	Unused contracts in `NGOLis.sol`, `NGOLisFactory.sol`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	-	7
NGOLisFactory.sol	-	6

Description

In the mentioned locations, contracts and libraries are imported but not used within the contracts.

Recommendation

We recommend removing unused imports to maintain codebase cleanliness.

Update

Client's response

Removed all unused imports, contacts from `NGOLis.sol` and `NGOLisFactory.sol`.

Oxorio's response

Fixed in `51f49e43dec8b70f71fb2016c442ff05b198a35b`.

I-22	Unused error in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis`	229

Description

In the contract `NGOLis`, the error `FeeError` is defined but not used in the code.

Recommendation

We recommend removing the definition of the `FeeError` to maintain codebase cleanliness.

Update

Client's response

Removed all unused errors from `NGOLis.sol`.

Oxorio's response

Fixed in `51f49e43dec8b70f71fb2016c442ff05b198a35b`.

I-23	Misleading variable name `wstAmount` instead of `stAmount` in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `claimWithdrawInWStEth`	763

Description

In the function `claimWithdrawInWStEth` of contract `NGOLis`, the value returned by the function `getStETHByWstETH` is assigned to the variable `wstAmount`, although the value represents `stEth` instead of `wstEth`, which might cause confusion:

```
uint256 wstAmount = wstETHSC.getStETHByWstETH(_amount);
```

Recommendation

We recommend renaming the variable `wstAmount` to correspond to the token `stEth`, such as `stAmount`.

Update

Client's response

Variable wstAmount was renamed to `stEthAmount`.

Oxorio's response

Fixed in `51f49e43dec8b70f71fb2016c442ff05b198a35b`.

I-24	Incorrect event parameter value `wstEth` instead of `stEth` in `NGOLis`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `claimWithdrawInWStEth`	776

Description

In the function `claimWithdrawInWStEth` of contract `NGOLis`, the parameter `_amount`, which represents the size of `wstEth` for the claim, is passed when emitting an event:

```
* @param _amount Amount of WStEth for claiming.
```

However, the event description specifies that the parameter represents `stEth`:

```
* @param _amount The amount of stETH claimed.
```

Recommendation

We recommend passing the documented value when emitting the event or updating the parameter description accordingly.

Update

Client's response

Changed documentation for the event. It describes that the amount passed to an event relies on `ETH` type.

Oxorio's response

Fixed in `51f49e43dec8b70f71fb2016c442ff05b198a35b`.

I-25	Inconsistent state variable naming style in `NGOLis`, `NGOLisFactory`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis`	397
NGOLis.sol	contract `NGOLis`	402
NGOLis.sol	contract `NGOLis`	407
NGOLis.sol	contract `NGOLis`	412
NGOLis.sol	contract `NGOLis`	417
NGOLis.sol	contract `NGOLis`	422
NGOLisFactory.sol	contract `NGOLisFactory`	41
NGOLisFactory.sol	contract `NGOLisFactory`	45
NGOLisFactory.sol	contract `NGOLisFactory`	49
NGOLisFactory.sol	contract `NGOLisFactory`	54

Description

In the mentioned locations, state variables have inconsistent naming styles and modifiers:

- In `NGOLis`, variables lack the `public` modifier despite not having leading underscores.
- In `NGOLisFactory`, `internal` variables are named inconsistently, some with leading underscores and some without.

Recommendation

We recommend refactoring variable names and applying appropriate modifiers to ensure consistent style and clarity in the code.

Update

Client's response

Based on the response and solidity style guide we have refactored variable naming style.
All private variables have leading underscores, while all public variables without leading underscores.

Oxorio's response

Fixed in `51f49e43dec8b70f71fb2016c442ff05b198a35b`.

I-26	Missing event emissions in setters in `NGOLis`, `NGOLisFactory`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `setOracle`	880
NGOLis.sol	contract `NGOLis` > function `setRewardsOwner`	889
NGOLis.sol	contract `NGOLis` > function `setUserBan`	898
NGOLisFactory.sol	contract `NGOLisFactory` > function `setImplementetion`	149

Description

In the mentioned locations, events are not emitted when state changes occur in setters.

For example, in the functions `setOracle` and `setRewardsOwner`, the addresses of `oracle` and `rewardOwner` are updated. At the same time, when `NGOLis` is created in the constructor of the `NGOLisFactory` contract, the `NGOCreated` event includes the `oracle` and `rewardOwner`.

This results in the absence of events when addresses are updated via setters, leading to untracked changes and outdated data in indexed tools like subgraph.

Recommendation

We recommend adding event emissions when setters are invoked.

Update

Client's response

Added event emissions in setters in NGOLis and NGOLisFactory:

- `setOracle`
- `setRewardsOwner`
- `setUserBan`
- `setImplementation` -> this function was renamed (previously setImplementetion).

Oxorio's response

Fixed in `51f49e43dec8b70f71fb2016c442ff05b198a35b`.

I-27	Missing parameter validation in `NGOLis`, `NGOLisFactory`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLis.sol	contract `NGOLis` > function `setUserBan`	897
NGOLisFactory.sol	contract `NGOLisFactory` > `constructor`	73
NGOLisFactory.sol	contract `NGOLisFactory` > function `setImplementetion`	149

Description

In the mentioned locations, parameter validation is missing:

- In the contracts `NGOLis` and `NGOLisFactory`, the setter functions `setUserBan` and `setImplementetion` do not validate for the zero address.
- In the constructor of the `NGOLisFactory` contract, parameter validation is absent, unlike the constructor of the `NGOLis` contract.

Recommendation

We recommend adding parameter validation at the specified locations.

Update

Client's response

Added validations in:
setImplementation -> NGOLisFactory
constructor -> NGOLisFactory
setUserBan -> NGOLis

Oxorio's response

Fixed in `51f49e43dec8b70f71fb2016c442ff05b198a35b`

I-28	Setting `msg.sender` as `owner` during deployment in `NGOLisFactory`
Severity	**INFO**
Status	• FIXED

Location

File	Location	Line
NGOLisFactory.sol	contract `NGOLisFactory` > `constructor`	72

Description

In the `constructor` of the `NGOLisFactory` contract, `msg.sender` is assigned as the `owner`:

```
) Ownable(msg.sender) {
```

Typically, the deployer role differs from the role of the user(s) maintaining the contracts. Consequently, the deployer would likely need to transfer ownership to another address.

Recommendation

We recommend specifying the `owner` address at deployment time and setting a multi-signature wallet, such as through a Safe Wallet.

Update

Client's response

Changed `msg.sender` to `_owner` variable.
Must be set up during function call.

Oxorio's response

Fixed in `51f49e43dec8b70f71fb2016c442ff05b198a35b`.

4. APPENDIX

4.1 SECURITY ASSESSMENT METHODOLOGY

Oxorio's smart contract security audit methodology is designed to ensure the security, reliability, and compliance of smart contracts throughout their development lifecycle. Our process integrates the Smart Contract Security Verification Standard (SCSVS) with our advanced techniques to address complex security challenges. For a detailed look at our approach, please refer to the full version of our methodology. Here is a concise overview of our auditing process:

1. Project Architecture Review

All necessary information about the smart contract is gathered, including its intended functionality and dependencies. This stage sets the foundation by reviewing documentation, business logic, and initial code analysis.

2. Vulnerability Assessment

This phase involves a deep dive into the smart contract's code to identify security vulnerabilities. Rigorous testing and review processes are applied to ensure robustness against potential attacks.

This stage is focused on identifying specific vulnerabilities within the smart contract code. It involves scanning and testing the code for known security weaknesses and patterns that could potentially be exploited by malicious actors.

3. Security Model Evaluation

The smart contract’s architecture is assessed to ensure it aligns with security best practices and does not introduce potential vulnerabilities. This includes reviewing how the contract integrates with external systems, its compliance with security best practices, and whether the overall design supports a secure operational environment.

This phase involves a analysis of the project's documentation, the consistency of business logic as documented versus implemented in the code, and any assumptions made during the design and development phases. It assesses if the contract's architectural design adequately addresses potential threats and integrates necessary security controls.

4. Cross-Verification by Multiple Auditors

Typically, the project is assessed by multiple auditors to ensure a diverse range of insights and thorough coverage. Findings from individual auditors are cross-checked to verify accuracy and completeness.

5. Report Consolidation

Findings from all auditors are consolidated into a single, comprehensive audit report. This report outlines potential vulnerabilities, areas for improvement, and an overall assessment of the smart contract's security posture.

6. Reaudit of Revised Submissions

Post-review modifications made by the client are reassessed to ensure that all previously identified issues have been adequately addressed. This stage helps validate the effectiveness of the fixes applied.

7. Final Audit Report Publication

The final version of the audit report is delivered to the client and published on Oxorio's official website. This report includes detailed findings, recommendations for improvement, and an executive summary of the smart contract's security status.

4.2 CODEBASE QUALITY ASSESSMENT REFERENCE

The tables below describe the codebase quality assessment categories and rating criteria used in this report.

Category	Description
Access Control	Evaluates the effectiveness of mechanisms controlling access to ensure only authorized entities can execute specific actions, critical for maintaining system integrity and preventing unauthorized use.
Arithmetic	Focuses on the correct implementation of arithmetic operations to prevent vulnerabilities like overflows and underflows, ensuring that mathematical operations are both logically and semantically accurate.
Complexity	Assesses code organization and function clarity to confirm that functions and modules are organized for ease of understanding and maintenance, thereby reducing unnecessary complexity and enhancing readability.
Data Validation	Assesses the robustness of input validation to prevent common vulnerabilities like overflow, invalid addresses, and other malicious input exploits.
Decentralization	Reviews the implementation of decentralized governance structures to mitigate insider threats and ensure effective risk management during contract upgrades.
Documentation	Reviews the comprehensiveness and clarity of code documentation to ensure that it provides adequate guidance for understanding, maintaining, and securely operating the codebase.
External Dependencies	Evaluates the extent to which the codebase depends on external protocols, oracles, or services. It identifies risks posed by these dependencies, such as compromised data integrity, cascading failures, or reliance on centralized entities. The assessment checks if these external integrations have appropriate fallback mechanisms or redundancy to mitigate risks and protect the protocol's functionality.
Error Handling	Reviews the methods used to handle exceptions and errors, ensuring that failures are managed gracefully and securely.
Logging and Monitoring	Evaluates the use of event auditing and logging to ensure effective tracking of critical system interactions and detect potential anomalies.
Low-Level Calls	Reviews the use of low-level constructs like inline assembly, raw `call` or `delegatecall`, ensuring they are justified, carefully implemented, and do not compromise contract security.

Category	Description
Testing and Verification	Reviews the implementation of unit tests and integration tests to verify that codebase has comprehensive test coverage and reliable mechanisms to catch potential issues.

4.2.1 Rating Criteria

Rating	Description
Excellent	The system is flawless and surpasses standard industry best practices.
Good	Only minor issues were detected; overall, the system adheres to established best practices.
Fair	Issues were identified that could potentially compromise system integrity.
Poor	Numerous issues were identified that compromise system integrity.
Absent	A critical component is absent, severely compromising system safety.
Not Applicable	This category does not apply to the current evaluation.

4.3 FINDINGS CLASSIFICATION REFERENCE

4.3.1 Severity Level Reference

The following severity levels were assigned to the issues described in the report:

Title	Description
CRITICAL	Issues that pose immediate and significant risks, potentially leading to asset theft, inaccessible funds, unauthorized transactions, or other substantial financial losses. These vulnerabilities represent serious flaws that could be exploited to compromise or control the entire contract. They require immediate attention and remediation to secure the system and prevent further exploitation.
MAJOR	Issues that could cause a significant failure in the contract's functionality, potentially necessitating manual intervention to modify or replace the contract. These vulnerabilities may result in data corruption, malfunctioning logic, or prolonged downtime, requiring substantial operational changes to restore normal performance. While these issues do not immediately lead to financial losses, they compromise the reliability and security of the contract, demanding prioritized attention and remediation.
WARNING	Issues that might disrupt the contract's intended logic, affecting its correct functioning or making it vulnerable to Denial of Service (DDoS) attacks. These problems may result in the unintended triggering of conditions, edge cases, or interactions that could degrade the user experience or impede specific operations. While they do not pose immediate critical risks, they could impact contract reliability and require attention to prevent future vulnerabilities or disruptions.
INFO	Issues that do not impact the security of the project but are reported to the client's team for improvement. They include recommendations related to code quality, gas optimization, and other minor adjustments that could enhance the project's overall performance and maintainability.

4.3.2 Status Level Reference

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

Title	Description
NEW	Waiting for the project team's feedback.

Title	Description
FIXED	Recommended fixes have been applied to the project code and the identified issue no longer affects the project's security.
ACKNOWLEDGED	The project team is aware of this finding and acknowledges the associated risks. This finding may affect the overall security of the project; however, based on the risk assessment, the team will decide whether to address it or leave it unchanged.
NO ISSUE	Finding does not affect the overall security of the project and does not violate the logic of its work.

4.4 ABOUT OXORIO

OXORIO is a blockchain security firm that specializes in smart contracts, zk-SNARK solutions, and security consulting. With a decade of blockchain development and five years in smart contract auditing, our expert team delivers premier security services for projects at any stage of maturity and development.

Since 2021, we've conducted key security audits for notable DeFi projects like Lido, 1Inch, Rarible, and deBridge, prioritizing excellence and long-term client relationships. Our co-founders, recognized by the Ethereum and Web3 Foundations, lead our continuous research to address new threats in the blockchain industry. Committed to the industry's trust and advancement, we contribute significantly to security standards and practices through our research and education work.

Our contacts:

- oxor.io
- ping@oxor.io
- Github
- Linkedin
- Twitter

THANK YOU FOR CHOOSING