Liquid Staking Token Security Review Report

January 9, 2025

Version 1.0

Email	Telephone
audits@extropy.io	+44 1865338228

Data Classification	Client Confidential
Client Name	Stakingverse
Document Title	Liquid Staking Token Security Review Report
Document Version	1.0
Author	Extropy Audit Team

Contents

1 Executive Summary **1**

2 Audit summary **2**
- 2.1 Audit scope . . . 2
- 2.2 Issues Summary . . . 2
- 2.3 Methodology . . . 2
- 2.4 Approach . . . 3

3 System Overview **4**
- 3.1 Vault . . . 4
- 3.2 Liquid Staking Token . . . 7
- 3.3 Liquid Staking Token AutoMint Extension . . . 9

4 Findings **10**
- 4.1 [HIGH] Code invariant can be broken . . . 10
- 4.2 [HIGH] Accounting can be compromised if the sLYXToken contract address is the deposit beneficiary . . . 12
- 4.3 [MEDIUM] Centralized roles in ‘Vault‘ . . . 13
- 4.4 [LOW] returned LYX from Beacon Chain are still accounted as ‘totalStaked‘ erroneously . . . 14
- 4.5 [LOW] Missing checks . . . 16
- 4.6 [LOW] Unverified implementation for ‘Vault‘ contract . . . 17
- 4.7 [INFO] Unnecessary check in ‘setFee()‘ . . . 17
- 4.8 [INFO] ‘feeRecipient‘ name and comment are misleading . . . 18
- 4.9 [INFO] Redundant modifiers . . . 19
- 4.10 [INFO] ‘LST‘ and ‘sLYX‘ are used interchangeably . . . 19
- 4.11 [INFO] Centralized functions do not emit events on state change . . . 20
- 4.12 [INFO] No upper bound for the loop within ‘registerValidators()‘ . . . 20
- 4.13 [INFO] For loop can revert earlier . . . 21
- 4.14 [INFO] Deposit will be blocked after ‘totalValidatorsRegistered‘ reaches the maximum value allowed . . . 22

5 Test Coverage **23**
- 5.1 Solidity tests . . . 23
- 5.2 Coverage . . . 25

A Disclaimers **26**
- A.1 Client Confidentiality . . . 26
- A.2 Proprietary Information . . . 26

B Slither results **27**
- B.1 LiquidStakingToken.sol . . . 27
- B.2 LiquidStakingTokenAutoMintExtension.sol . . . 27
- B.3 Vault.sol . . . 28

1 | Executive Summary

Extropy was contracted to conduct an initial code review and vulnerability assessment of the Liquid Staking Token protocol.

The Liquid Staking Token contract (shortly LST contract) is a fungible token compliant to the LSP7 standard on the LUKSO network and allows users to receive "**sLYX**" liquid tokens after having staked their LYX, the native currency of the LUKSO network, in a Vault: while they are receiving rewards through the Vault, which stakes them on the Beacon Chain, they can use the minted sLYX on other De-Fi protocols.

Users after having staked their LYX in the Vault and being entitled to a certain amount of shares depending on the deposit size, are able to transfer the ownership of those shares to the LST contract which will mint sLYX to them in return.

Burning sLYX allows the LST contract to return the ownership of the shares to their respective initial owners. Later they can use those shares to withdraw from the Vault the initial LYX deposit plus the accrued rewards.

There is also another contract called "LiquidStakingTokenAutoMintExtension" which is compliant to the LSP17 standard and allows users to perform in one single call all the operations mentioned above: depositing LYX to the Vault, transferring the shares to LST contract and mint sLYX.

The code of the components analyzed in this document is very well organized. Section 4 details the findings, and where possible we have given recommendations for their resolution.

2 | Audit summary

This audit, conducted from October 21 2024 to November 1 2024, employed a comprehensive approach combining manual review and automated review methods. Our examination aimed to ensure the robustness and security of the Liquid Staking Token codebase.

- The audit was performed on commit 89b7966 (main branch) of **pool-contracts** repository.

2.1 | Audit scope

The following contracts were audited:

Contract	LoC
LiquidStakingToken.sol	147
LiquidStakingTokenAutoMintExtension.sol	62
UniversalPage-contracts/contracts/src/pool/Vault.sol	512
Total	**721**

2.2 | Issues Summary

ID	Finding	Status
4.1	[HIGH] Code invariant can be broken	Resolved
4.2	[HIGH] Accounting can be compromised if the sLYXToken contract address is the deposit beneficiary	Resolved
4.3	[MEDIUM] Centralized roles in ‘Vault‘	Mitigated
4.4	[LOW] returned LYX from Beacon Chain are still accounted as ‘totalStaked‘ erroneously	Acknowledged
4.5	[LOW] Missing checks	Partially Resolved
4.6	[LOW] Unverified implementation for ‘Vault‘ contract	Resolved
4.7	[INFO] Unnecessary check in ‘setFee()‘	Resolved
4.8	[INFO] ‘feeRecipient‘ name and comment are misleading	Partially Resolved
4.9	[INFO] Redundant modifiers	Resolved
4.10	[INFO] ‘LST‘ and ‘sLYX‘ are used interchangeably	Resolved
4.11	[INFO] Centralized functions do not emit events on state change	Resolved
4.12	[INFO] No upper bound for the loop within ‘registerValidators()‘	Acknowledged
4.13	[INFO] For loop can revert earlier	Acknowledged
4.14	[INFO] Deposit will be blocked after ‘totalValidatorsRegistered‘ reaches the maximum value allowed	Acknowledged

2.3 | Methodology

2.3.1 | Risk Rating

The risk rating given for issues follows the standard approach of the OWASP Foundation. We combine two factors :

- Likelihood of exploit
- Impact of Exploit

The Categories we use are *Critical*, *High*, *Medium*, *Low* and *Informational* These categories may not align with the categories used by other companies.
The informational category is used to contain suggestions for optimisation (where this is not seen as causing significant impact), or for alternative design or best practices.

2.4 | Approach

The project was assessed mainly by code inspection, with auditors working independently or together, looking for possible exploits.
Tests were written were possible to validate the issues found.
Static analysis tools were also used, and summary results are attached to this report. These tools are seen as an adjunct to code inspection.

3 | System Overview

As detailed in the Executive Summary section of the document, the protocol allows users to stake LYX in a Vault and receive sLYX liquid tokens in return to use across other De-Fi protocol if they transfer the ownership of the deposit shares to the LST contract.
The codebase is made up of three main components:

- Vault
- Liquid Staking Token
- Liquid Staking Token AutoMint Extension

3.1 | Vault

The Vault contract it's a fork of UniversalPage's Vault and it's an upgradeable contract:

- The proxy is deployed at address 0x9F49a95b0c3c9e2A6c77a16C177928294c0F6F04
- The current implementation (unverified when writing) is deployed at 0x2Cb02ef26aDDAB15686ed634d70699ab64F195f4 on the LUKSO chain

The Vault is the contract where users deposit LYX and it will accrue rewards on behalf of users by staking funds on the Beacon Chain. After depositing, users are able to transfer the deposit shares to the LST contract to mint sLYX in return if they want.

Vault is an upgradeable and pausable contract and has three roles which can interact with specific functions:

- **owner**: able to pause and unpause the contract
- **operator**: able to set oracle addresses, fee amounts, deposit limits and address allowed to deposit if the Vault is put in a restricted status
- **oracle**: able to register new validators for the Beacon Chain staking and call a crucial contract function named "rebalance()"

Vault can be set by the operator address in a restricted status which means only a certain list of whitelisted addresses can deposit. Otherwise all users are allowed to deposit through the deposit() function and being entitled of a certain amount of shares in the _shares mapping for that specific account.

Users deposits can be moved by the oracle addresses to the deposit contract of the Beacon Chain while registering new validators. More details on that process are given in next section.
Vault withdrawals are possible for users up to the value stored in a variable called 'totalUnstaked': the excess is counted as pending withdrawal and can be claimed later through 'claim()'.
Vault keeps track of Vault balances using the following values:

- **totalStaked**: refers to the total amount of active stake in wei on beacon chain. It's increased by 32 LYX every time a new validator is registered and updated when the oracle calls rebalance().
- **totalUnstaked**: refers to the total amount of inactive stake in wei on execution layer. It's increased by the amount the users deposit to the contract and decreased both when a user withdraws from the Vault and decreased by 32 LYX every time a new validator is registered. It's also updated when the oracle calls rebalance().

- **totalClaimable**: refers to the total amount of wei available to be claimed for existing pending withdrawals. It's updated when the oracle calls rebalance() and decreased when the user calls claim().
- **totalPendingWithdrawal**: refers to the total amount of pending withdrawals in wei. When a user withdraws from the Vault and all the amount is not available that counter is increased (if totalUnstaked is not enough to cover the amount being withdrawn). That counter is decreased when the user calls claim().

3.1.1 | Beacon Chain withdrawals

Vault is able to accrue rewards by staking users funds to the Beacon Chain.

Specifically through ‘registerValidator()‘ 32e18 LYX are sent from the Vault to the deposit contract specified in the constructor on Vault deployment, increasing ‘totalStaked‘ counter and decreasing ‘totalUnstaked‘.

Within ‘registerValidator()‘ a parameter called ‘withdrawCredentials‘ must be specified, computed as the concatenation of the following terms:

- **0x01**: means that accrued staking rewards are automatically distributed to the **withdraw recipient address**
- 11 empty bytes (**0x00**)
- the **withdraw recipient address**. In this case the Vault contract itself.

Two types of withdraws are possible from the Beacon Chain:

- **partial**: only accrued staking rewards (distributed automatically)
- **full**: 32e18 LYX plus other eventual accrued reward not already distributed

It's important to note that funds withdrawn from the Beacon Chain to a contract, do not trigger its receive() or fallback() functions. The funds are directly transferred to the contract, similar to a force-send and if the contract needs to handle those new funds it must explicitly expose methods for that. In the Vault contract the ‘rebalance()‘ function takes care of that. More details in the next section.

3.1.2 | The rebalance() function

One of the most relevant functions of the Vault contract is the ‘rebalance()‘ function: it takes care mainly of updating ‘totalClaimable‘, ‘totalUnstaked‘ and ‘totalStaked‘ state variables. It keeps the internal accounting coherent with the new funds the contract may have received such as rewards or full exits from the Beacon Chain. This function can be triggered by any address with the ‘oracle‘ role.

Function code can be divided into five sections:

1. **Computing the number of completed withdrawals**. This first section firstly saves in a local variable called ‘balance‘ the actual balance in LYX of the contract. This line will account total funds in the Vault made up of user deposits, funds eventually returned from the beacon chain (both rewards for partial exits or 32 LYX + rewards for full exits), funds transferred to Vault by mistake by users or sent with selfdestruct: they will be accounted as rewards. Note: even if selfdestruct deprecation is still in effect, contracts can still force sending ETH to vault when writing without triggering the receive() function.

Then ‘pendingWithdrawal‘ is computed as the difference between the amount of the pending withdrawals and the amount already reserved previously in ‘totalClaimable‘ to be immediately claimed by users.

Finally ‘completedWithdrawal‘ computes the amount that was withdrawn from the Beacon Chain as the minimum between two values:

- the current balance minus ‘totalFees‘ eventually computed in previous calls, ‘totalUnstaked‘ which are user deposits to the Vault and ‘totalClaimable‘ . The result is divided by 32e18 to compute the number of full withdrawals.
- ‘pendingWithdrawal‘ divided 32e18 plus 1 or 0 depending on whether the result of ‘pending-Withdrawal‘ modulo ‘32e18‘ has reminder or not respectively

Once computed the minimum number of withdrawals that value is multiplied by 32e18 again to compute the amount in LYX. Computing the minimum value means ensuring the minimum number of full withdrawals while other eventual partial withdrawals will be accounted later.

```
function rebalance() external onlyOracle whenNotPaused {
uint256 balance = address(this).balance;

uint256 pendingWithdrawal = totalPendingWithdrawal - totalClaimable;
uint256 completedWithdrawal = Math.min(
    (balance - totalFees - totalUnstaked - totalClaimable) /
    ↪  DEPOSIT_AMOUNT,
    pendingWithdrawal / DEPOSIT_AMOUNT
        + (pendingWithdrawal % DEPOSIT_AMOUNT == 0 ? 0 : 1)
) * DEPOSIT_AMOUNT;
...
```

2. **Updating local variables: staked, unstaked, claimable**. Specifically:

- ‘staked‘ is updated with the old ‘totalStaked‘ minus the amount withdrawn from the Beacon Chain
- ‘unstaked‘ is updated with the new contract balance minus ‘totalFees‘, ‘totalClaimable‘ and ‘completedWithdrawal‘.
- ‘claimable‘ is updated with the old ‘totalClaimable‘ plus ‘completedWithdrawal‘. This is why it was not added in ‘unstaked‘: it’s an amount reserved for being claimed through ‘claim()‘

```
...
uint256 staked = totalStaked - completedWithdrawal;
uint256 unstaked = balance - totalFees - totalClaimable -
↪  completedWithdrawal;
uint256 claimable = totalClaimable + completedWithdrawal;
...
```

3. **Adjust local variables accounting for partial withdrawals**. If the new amount stored in ‘claimable‘ is greater than the total amount of user withdrawals in pending there is no need to have this surplus: ‘claimable‘ is capped to ‘totalPendingWithdrawal‘ and the excess is added to ‘unstaked‘

```
...
uint256 partialWithdrawal = 0;
```

```
        if (claimable > totalPendingWithdrawal) {
            partialWithdrawal = claimable - totalPendingWithdrawal;
            unstaked += partialWithdrawal;
            claimable = totalPendingWithdrawal;
        }
        ...
```

4. **Subtracting fees from rewards**. If the value of ‘unstaked‘ (without the ‘partialWithdrawal‘ amount just added) is greater than the old value stored in ‘totalUnstaked‘ means that the balance of the contract increased despite the amount reserved again for ‘totalFees‘ and ‘totalClaimable‘. An increasing balance means that rewards were sent to contract by the Beacon Chain or that funds were transferred to Vault by mistake by users or sent with selfdestruct: they will be accounted as rewards.

   Being rewards, a fee is taken on top of them which will be later claimed through ‘claimFees()‘ by the ‘feeRecipient‘. ‘totalFees‘ and ‘unstaked‘ are updated accordingly.

```
        ...
        if (unstaked - partialWithdrawal > totalUnstaked) {
            uint256 rewards = unstaked - partialWithdrawal - totalUnstaked;
            uint256 feeAmount = Math.mulDiv(rewards, fee, _FEE_BASIS);
            emit RewardsDistributed(totalStaked + totalUnstaked, rewards,
             ↪  feeAmount);
            totalFees += feeAmount;
            unstaked -= feeAmount;
        }
        ...
```

5. **Updating contract state**. The state variables ‘totalClaimable‘, ‘totalUnstaked‘ and ‘totalStaked‘ are updated with the local values of ‘claimable‘, ‘unstaked‘ and ‘staked‘.

```
        ...
        totalClaimable = claimable;
        totalUnstaked = unstaked;
        totalStaked = staked;
        ...
```

3.2 | Liquid Staking Token

The Liquid Staking Token contract (LST contract) is based on the ‘LSP7‘ Token standard for fungible tokens. An overview to the standard is given in the next section.
A user is able to mint sLYX tokens by invoking a function named ‘transferStake(address to, uint256 amount, bytes calldata data)‘ on the Vault contract specifying:

- the LST contract itself as the ‘to‘ parameter:
- the ‘amount‘ of LST to mint which should be lower than the amount of LYX staked by that user in the vault
- additional ‘data‘

In this way the vault will set the LST contract as the owner of that amount of shares and will trigger 'onVaultStakeReceived()' on the LST contract for the minting process.

sLYX can be transferred as normal tokens. If a user called Alice transfers its sLYX tokens to another user Bob, she is essentially transferring her stake position in the Vault to Bob. Users can also transfer part of their sLYX balance and therefore transferring the ownership only of a portion of their stake.

As long as the users hold sLYX tokens, they cannot unstake the deposit from the Vault: in order to unstake they must burn the amount of sLYX from which the amount of shares to unlock is computed on. When a burn operation of sLYX is conducted, a transfer to the address zero is performed and the function '_afterTokenTransfer()' takes care of transferring the stake (the shares in Vault contract) again from the LST contract to the sLYX token holder who is burning them.

Two relevant function exist in the LST contract which use the shares and balance values from the Vault contract to calculate how much sLYX are backed from LYX and vice versa:

- **getNativeTokenValue()**: determines the value of sLYX in terms of the native token balance (LYX) held in the staking Vault. Takes in 'sLyxAmount' (the sLYX token amount) as input and return the equivalent LYX amount.
- **getLiquidStakingTokenValue()**: calculates the number of sLYX that would be backed by a given amount of LYX. Takes in 'sLyxAmount' as input and return the equivalent LYX amount.

3.2.1 | LSP7 Token Standard

On the LUKSO network, 'LSP7' is the standard for fungible assets; however it also can be used for non-fungible-like assets under certain circumstances. Its main goal is to overcome Ethereum standards limitations by:

- Giving the possibility to the token deployer of deciding **whether the token should be divisible or non-divisible**. Namely divisible assets have decimals (up to 18), thus is possible to transfer fractional token amounts. Non-divisible assets instead doesn't have decimals (decimals() returns 0), thus you cannot transfer less than 1 token: only whole token units. This means that each non-divisible LSP7 token can represent, for example, tickets of equal value.
- **Allowing unlimited asset metadata** via LSP4-DigitalAssetMetadata standard. ERC20 and ERC721 only comes with metadata such as name, symbol and tokenURI. The goal is to provide flexible and extensible storage to attach any other metadata to any token. The combination between LSP7 token divisibility and metadata flexibility enable to the developer new combinations as shown in this table here.
- **Enabling token hooks** between sender/recipient via LSP1-UniversalReceiver.After a LSP7 token transfer is executed and sender/recipient balances updated the token itself inform them of the transfer using hooks, only if they are contracts and not EOAs. Sender/Recipient contracts can react on the incoming transfer by accepting or rejecting the tokens, or implementing a custom logic to run on each transfer with the help of LSP1-UniversalReceiverDelegate. The LSP7 token contract is able to notify sender and recipient using '_notifyTokenSender()' and '_notifyTokenReceiver()' functions only if they implement LSP1 standard, of course.
- **Allowing forced transfers** via force boolean parameter in the 'transfer()' and 'mint()' functions. In general, addresses that are expected to interact with the LUKSO network are smart contract accounts; EOAs can receive tokens, but should be used mainly to control smart contracts, not

to interact on the network or hold tokens. For this reason there is a flag called 'force' that can be set during a transfer/mint operations. If false, the transfer will only pass if the recipient is a smart contract that implements the LSP1-UniversalReceiver standard: if the recipient is an EOA or a smart contract not implementing LSP1, the transfer will fail. Instead if the flag is set to true the recipient can be either an EOA, a smart contract that implements LSP1 or a smart contract that doesn't implement LSP1.

A technical reference containing LSP7 function descriptions can be found here, while the standard implementation is available in the lukso-network repository itself.

3.3 | Liquid Staking Token AutoMint Extension

The Liquid Staking Token AutoMint Extension contract is based on the LSP17 extension.

The standard increases smart contract flexibility and upgradeability on the LUKSO blockchain. It allows a main contract to be associated with several extension contracts each of them implementing a certain functionality.

The core logic of LSP17 lies in its forwarding strategy: when the main contract receives a call for an undefined function, it reroutes this to the appropriate Extension Contract. This routing is made possible through a specialized fallback function, ensuring that Extension Contracts can access the original call details, including the initiator and the transaction value appended to the calldata.

The Liquid Staking Token AutoMint Extension contract is an extension for the LST contract. When users call "deposit()" on the LST contract (which not exist) that call is forwarded to the extension contract and performs several operations in one transaction: deposit LYX to the Vault, transfer the ownership of the received shares to LST contract, mint sLYX and transfer those sLYX from the extension contract (that received them) to the beneficiary address.

As noted earlier, there is no function 'deposit()' on Liquid Staking Token contract: it forwards the call to the extension contract which owns that logic using the fallback function inherited from LSP17 standard.

The forwarding process works fine only if previously an extension address was set for a specific function selector using the ERC725Y storage (in this case the function selector of 'deposit(address)' is '0xf340fa01').

The ERC725Y storage is a mapping from bytes32 to bytes and allows to store a 'dataValue' (the extension address) for a specific 'dataKey', which is built using the function selector. More info here.

4 | Findings

4.1 | [HIGH] Code invariant can be broken

- **Location(s):** LiquidStakingToken.sol#103, LiquidStakingToken.sol#117
- **Description:** Within `LiquidStakingToken.sol` the functions `getNativeTokenValue()` and `getLiquidStakingTokenValue()` makes use of `sharesOf()` to compute the total amount of shares held by the LiquidStakingToken contract. In both functions this is achieved using the following line of code:

```
// Get the total number of shares this LST contract holds (= equivalent
// to total number of LST tokens minted).
uint256 totalSLYXMinted = stakingVault.sharesOf(address(this));
```

As from the comment it's assumed that the number of shares held by the LiquidStakingToken contract is exactly equivalent to the total amount of `sLYX` minted to the LiquidStakingToken contract when a stake is transferred to it by a user using `Vault.transferStake()` triggering `LiquidStakingToken.onVaultStakeReceived()`.

This invariant is true, but can easily broken by a malicious actor by sending some few gwei to the Vault contract using `selfdestruct` and so bypassing `receive()` and the call to `deposit(msg.sender)`. In this way the amount minted as sLYX in transferStake() does not reflect the number of shares added to the mapping `_shares[LiquidStakingTokenContract]` due to a wrong value computed by `_toShares()`.

Let's suppose we have the following scenario:

- an entity called `depositor1` makes a deposit() of 70 LYX
- an entity called `depositor2` makes a deposit() of 2 LYX
- `depositor1` calls `transfersStake()` to the LST contract with an amount of 20e18, receiving 20e18 sLYX. At this point the invariant `sharesOf(liquidStakingTokenContract) == totalSLYXMinted` is still valid
- a malicious user creates a dummy contract, funds it with some gwei, calls selfdestruct on it forwarding those funds to the Vault contract bypassing `receive()`
- `rebalance()` is called on the Vault
- `depositor2` calls `transfersStake()` to the LST contract with an amount of 1e18, receiving 1e18 sLYX. At this point the invariant `sharesOf(liquidStakingTokenContract) == totalSLYXMinted` is not valid anymore

A test is provided for that scenario:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.22;

import {Test, console} from "forge-std/Test.sol";
import {Vault, IVault} from "UniversalPage-contracts/src/pool/Vault.sol";
import {LiquidStakingToken} from "../src/LiquidStakingToken.sol";
import {LiquidStakingTokenAutoMintExtension} from
                    "../src/LiquidStakingTokenAutoMintExtension.sol";
import {LiquidStakingTokenBaseTest} from "./base/LiquidStakingTokenBaseTest.t.sol";
import {MockDepositContract} from "./mocks/MockDepositContract.sol";

contract issue15 is LiquidStakingTokenBaseTest{

    function setUp() public {
        _setUpLiquidStakingToken({setDepositExtension: true});
    }

    function test_Issue15() public makeInitialDeposit{
        //deposit 70 LYX
        address depositor1 = makeAddr("depositor1");
        vm.deal(depositor1, 70 ether );
        vm.prank(depositor1);
        vault.deposit{value: 70 ether}(depositor1);

        //deposit other 2 LYX
        address depositor2 = makeAddr("depositor2");
        vm.deal(depositor2, 2 ether );
        vm.prank(depositor2);
        vault.deposit{value: 2 ether}(depositor2);

        //depositor1 transfers 20 LYX to LST contract
        vm.prank(depositor1);
        vault.transferStake(address(liquidStakingToken), 20 ether, "");

        //the invariant `sharesOf(liquidStakingTokenContract) == totalsLYXMinted`
          is still valid
        assertEq(vault.sharesOf(address(liquidStakingToken)),
                  liquidStakingToken.totalSupply());

        // malicious user can force sending few LYX to the Vault using selfdestruct
        // not triggering receive() and no triggering deposit() and breaking the invariant
        vm.deal(address(vault), address(vault).balance + 200000 gwei);
        vm.prank(vaultOracle);
        vault.rebalance()

        //depositor2 transfers 1 LYX to LST contract
        vm.prank(depositor2);
        vault.transferStake(address(liquidStakingToken), 1 ether, "");

        // INVARIANT IS BROKEN
        assertEq(vault.sharesOf(address(liquidStakingToken)),
                    liquidStakingToken.totalSupply());

    }
}
```

- **Recommendation:** Consider computing `getNativeTokenValue()` and `getLiquidStakingTokenValue()` using the actual number of sLYX minted through `_existingTokens` which can be read using

LiquidStakingToken.totalSupply() function inherited from LSP7DigitalAssetInitAbstract contract.

- **Status:** Resolved.
- **Updates:**
  - [StakingVerse, 06/01/2025]: "This finding was fixed by making the SLYX token contract now uses Vault's shares (instead of staked LYX balance) to compute the amount of SLYX tokens to be minted. It was considered that the change of minting sLYX amount as shares instead of amount of transferred LYX deposited was an acceptable change, since minted sLYX are only a representation of how many shares the user is entitled. The PoC test that breaks the invariant was added to the invariant test suite and now pass after the fix mentioned above was implemented."
  - [Extropy, 06/01/2025]: Fixed in commit e6a9208[1]. Logic changed to use shares in commit 0ee1580[2]

4.2 [HIGH] Accounting can be compromised if the sLYXToken contract address is the deposit beneficiary

- **Location(s):** Vault.sol#291
- **Description:** Within Vault.sol the deposit() function allows users to deposit LYX tokens to the vault, which results in increasing the value stored in _shares[] mapping for the beneficiary address in input depending on the deposit size.

```
function deposit(address beneficiary) public payable override whenNotPaused {
    ...
   uint256 shares = _toShares(amount);
   if (shares == 0) {
      revert InvalidAmount(amount);
   }
    ...
    _shares[beneficiary] += shares;
    ...
 }
```

  However, a malicious user can call deposit() specifying the sLYX token contract address as the beneficiary compromising the accounting. Indeed since it is expected that the _shares[sLYXTokenAddress] only increases via transferStake(), this creates a scenario were the shares held by the SLYXToken contract are greater than its totalSupply() (namely the minted sLYX). This implies that when users burn their sLYX tokens, they get back a LYX amount different than their initial deposit + accumulated rewards.

- **Recommendation:** Consider ensuring that _shares[sLYXTokenAddress] only increases via transferStake() by denying deposits made to the Vault with deposit() if the beneficiary address specified as the input is the sLYX token contract address.
- **Status:** Resolved.
- **Updates:**

[1]https://github.com/Stakingverse/pool-contracts/commit/e6a9208e0f0ed57de61c13705bb7a47c6fed7fc8
[2]https://github.com/Stakingverse/pool-contracts/commit/0ee1580635fa9a16d0c8fcc44b33ecd9aa7b3a67

- [Stakingverse, 06/01/2025]: "A callback `onVaultStakedReceived` was added inside the `deposit(...)` function of the Vault. This allows a user to deposit and mint sLYX tokens immediately, resolving this issue. Some additional tests were added to ensure that both methods to mint sLYX (first one via `deposit` + `transferStake` to SLYXToken contract, second via `deposit` passing SLYXToken address as beneficiary parameter to the function). As a result, the contract `LiquidStakingTokenAutoMintExtension` - initially created to allow this feature of depositing LYX in the Vault + minting sLYX immediately on deposit - was removed from the codebase, as deemed unecessary since the addition of the callback in the `deposit` function now enables that."
- [Extropy, 06/01/2025]: Fixed in commit 90375e2[3]. Two methods (first one via `deposit` + `transferStake` to SLYXToken contract, second via `deposit` passing SLYXToken address as beneficiary parameter to the function) act the same also because of in `onVaultStakeReceived()` in the SLYXToken contract the amount to be minted is computed again calculating the shares using the new vault balances (`stakingVault.totalShares()` and `stakingVault.totalAssets()`) after the deposit.

4.3 | [MEDIUM] Centralized roles in 'Vault'

- **Location(s):** Vault.sol
- **Description:** The contract `Vault` contains two centralized roles:
  - `operator` is the role associated to address stored in the `operator` state variable on deployment. If that address is compromised there is no way to change it since the function `setOperator()` can only be called by the `operator` itself:
    - In the scenario where the private key associated with `operator` is lost, all the functions marked with `onlyOperator()` modifier (including `setOperator()`, `setFee()`, `setFeeRecipient()`, `setDepositLimit()`, `enableOracle()`, `allowList()` and `setRestricted()`) cannot be executed.
    - Instead, in the scenario where the private key is compromised, a malicious entity can leverage this role and perform dangerous actions against the Vault contract.

```
function _setOperator(address newOperator) private {
    if (newOperator == address(0)) {
        revert InvalidAddress(newOperator);
    }
    operator = newOperator;
}
```

  - `oracle` is the role associated to addresses in the `_oracles` mapping with a value set to `true`. Those addresses are the only ones able to call `rebalance()` and `registerValidator()`. Particularly if an oracle address acts maliciously (before an operator disables it through `enableOperator()`) it can deposit all the `totalStaked` amount available to the Beacon Chain using multiple `registerValidator()` calls and specifying a wrong or malicious `pubkey`.
    - This is not a real issue for partial withdrawals since rewards are automatically distributed to the address specified in `withdrawCredentials` on deposit.
    - Instead for full withdrawals (32 LYX principal stake + rewards) this may cause problems. Indeed making a full withdraw is a process that starts with an "exit request" message

[3] https://github.com/Stakingverse/pool-contracts/commit/90375e246990f0e5bd912efc0145a708303f5750

signed from the private key associated with the `pubkey` specified in `withdrawCredentials`. If the message is not signed, deposited funds may be locked and the full withdraw will not be possible: only rewards can be collected.

- **Recommendation:** Consider to always monitoring `operator` and `oracle` roles within Vault contract or implement a multi-sig solution for those privileged roles.
- **Status:** Mitigated.
- **Updates:**
  - `[StakingVerse, 06/01/2025]`: "The event `OperatorChanged` was added to resolve finding 4.10 (`Centralized functions do not emit events on state change`), which enables to monitor if the operator changes and mitigate the issue according to the recommendation provided."
  - `[Extropy, 06/01/2025]`: Emitting an event when the `operator` changes is a good way to monitor its updates over time. However this does not resolve the issues described in the finding associated with `operator` private key being lost or compromised. For the `oracle` role no changes were made instead.

4.4 | [LOW] returned LYX from Beacon Chain are still accounted as 'totalStaked' erroneously

- **Location(s):** Vault.sol#399-442
- **Description:** Within `Vault.sol` the function `rebalance()` updates several storage variables considering the amounts withdrawn from the Beacon Chain. For that it computes `completedWithdrawals` as the minimum between the Vault balance (minus values already accounted for fees or to be claimed) and the number of `pendingWithdrawals` (eventually plus one if there are partial withdrawals).

```
uint256 completedWithdrawal = Math.min(
    (balance - totalFees - totalUnstaked - totalClaimable) / DEPOSIT_AMOUNT,
    pendingWithdrawal / DEPOSIT_AMOUNT
        + (pendingWithdrawal % DEPOSIT_AMOUNT == 0 ? 0 : 1)
) * DEPOSIT_AMOUNT;
```

However a problem may arise in the accountings in a scenario where two full withdrawals happens from the beacon chain and the difference between `totalPendingWithdrawal` and `totalClaimable` is lower than `DEPOSIT_AMOUNT`.

Let's suppose the following scenario:

- an entity called `depositor` makes a `deposit()` of 70 LYX.
- an address marked as oracle calls `registerValidator()` two times.
- `depositor` calls withdraw() trying to withdraw part of his initial deposit. For example 30 LYX of the deposited 70 LYX.
- funds are returned from the Beacon Chain: the 64 staked LYX plus, as an example, 1 LYX reward
- supposing no fees are set, calling `rebalance()` will change the contract state causing `totalStaked` to be equal to 32e18 instead of 0

A test is provided for that scenario:

```solidity
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.22;

import {Test, console} from "forge-std/Test.sol";
import {Vault, IVault} from "UniversalPage-contracts/src/pool/Vault.sol";
import {LiquidStakingToken} from "../src/LiquidStakingToken.sol";
import {LiquidStakingTokenAutoMintExtension} from
         "../src/LiquidStakingTokenAutoMintExtension.sol";
import {LiquidStakingTokenBaseTest} from "./base/LiquidStakingTokenBaseTest.t.sol";
import {MockDepositContract} from "./mocks/MockDepositContract.sol";

contract issue10 is LiquidStakingTokenBaseTest{

    function setUp() public {
        _setUpLiquidStakingToken({setDepositExtension: true});
    }

    function test_Issue10() public makeInitialDeposit{
        //zeroing fees for simplicity
        vm.prank(vaultOperator);
        vault.setFee(0);

        //deposit 70 LYX
        address depositor = makeAddr("depositor");
        vm.deal(depositor, 70 ether );
        vm.prank(depositor);
        vault.deposit{value: 70 ether}(depositor);
        assertEq(address(vault).balance, _VAULT_INITIAL_DEPOSIT + 70 ether);
        assertEq(vault.totalUnstaked(), _VAULT_INITIAL_DEPOSIT + 70 ether);
        assertEq(vault.totalUnstaked(), vault.totalShares());
        assertEq(depositor.balance, 0);

        // register 2 validators
        vm.startPrank(vaultOracle);
        vault.registerValidator(hex"1234", hex"5678", bytes32(0));
        vault.registerValidator(hex"8745", hex"5678", bytes32(0));
        vm.stopPrank();
        assertEq(address(vault).balance, _VAULT_INITIAL_DEPOSIT + 6 ether);
        assertEq(vault.totalUnstaked(), _VAULT_INITIAL_DEPOSIT + 6 ether);
        assertEq(vault.totalStaked(), 64 ether);

        // withdraw 30 LYX out of 70
        vm.prank(depositor);
        vault.withdraw(30 ether, depositor);
        assertEq(address(vault).balance, 0);
        assertEq(vault.totalUnstaked(), 0);
        assertEq(vault.totalPendingWithdrawal(), 24 ether - _VAULT_INITIAL_DEPOSIT );
        assertEq(depositor.balance, 6 ether +  _VAULT_INITIAL_DEPOSIT);
        assertEq(vault.totalShares(), 40 ether +  _VAULT_INITIAL_DEPOSIT);
```

```
            // funds are returned from the Beacon Chain
            // 64 LYX + 1 LYX reward
            vm.deal(address(vault), address(vault).balance + 65 ether);
            assertEq(address(vault).balance, 65 ether);

            // calling rebalance WRONGLY keeps totalStaked = 32e18
            vm.startPrank(vaultOracle);
            vault.rebalance();
            assertEq(vault.totalStaked(), 0); // it's still 32 ether
            vm.stopPrank();
        }
    }
```

- **Recommendation:** Consider ensuring `totalStaked` is updated accordingly to reflect the amount of LYX currently staked on the Beacon Chain
- **Status:** Acknowledged.
- **Updates:**
  - `[StakingVerse, 20/11/2024]`: "This finding is not a feasible scenario. It is only possible if there is a bug in oracle code which would request to submit more withdrawals than required. If a user requested to withdraw 30 LYX, oracle would not exit 2 validators (64 LYX). It is acceptable risk, as oracle is under control of vault owner. Thus, worst case scenario, is that extra 32 LYX will go to unstaked amount, and LYX is distributed across all stakers, for exception of a fee which goes to vault owner."

4.5 | [LOW] Missing checks

- **Location(s):** Vault.sol#134, LiquidStakingToken.sol#63, LiquidStakingTokenAutoMintExtension.sol#38, LiquidStakingTokenAutoMintExtension.sol#39
- **Description:** Within the codebase some checks are missing:
  - Within the `initialize()` function of `Vault.sol` the input `owner_` is not checked to not be the zero address since this is not done within `OwnableUnset._setOwner()` code
  - Within the `initialize()` function of `LiquidStakingToken.sol` the input `stakingVault_` is not checked to not be the zero address before being used
  - Within `LiquidStakingTokenAutoMintExtension.sol` constructor the input `vaultAddress_` is not checked to not be the zero address before being used
  - Within `LiquidStakingTokenAutoMintExtension.sol` constructor the input `liquidStakingToken_` is not checked to not be the zero address before being used
- **Recommendation:** Consider ensuring that the above checks have been carried out to avoid undesirable behaviours of the code.
- **Status:** Partially resolved.
- **Updates:**
  - `[Extropy, 06/01/2025]`: Client fixed in commit 71dd845[4] all the occurrences except for Vault.sol#134

[4]https://github.com/Stakingverse/pool-contracts/commit/71dd8458955cf49f5ee1fae913add5dcb34f3358

4.6 | [LOW] Unverified implementation for 'Vault' contract

- **Location(s):** -
- **Description:** `Vault.sol` is an upgradeable contract made of:

  - proxy, deployed at 0x9F49a95b0c3c9e2A6c77a16C177928294c0F6F04[5]
  - implementation deployed at 0x2Cb02ef26aDDAB15686ed634d70699ab64F195f4[6]

  The way it works is that users interact with the proxy which forwards the calls to the current implementation. Implementation contract can be changed in the future by the admin, which currently is 0xe460f0cb1227399b129715895157e8cb443b269e[7].

  However two problems arise: 1. The current implementation code is not verified on chain. It means that there is no way to check that the implementation behaves exactly as the `Vault.sol` contract available here[8]. 2. The admin is able to change the implementation contract at any time updating the logic and leading to possible dangerous consequences.

- **Recommendation:** Consider ensuring that current implementation code is verified on-chain and aligned with `Vault.sol` contract available here[9]. Furthermore always monitor logic future updates.
- **Status:** Unresolved.
- **Updates:**

  - `[StakingVerse, 06/01/2025]`: "the source code of the Vault contract deployed at address 0x2Cb02ef26aDDAB15686ed634d70699ab64F195f4[10] has been verified. For any future upgrade of the Vault the proxy, the new implementation contract will be verified on chain to ensure users"

4.7 | [INFO] Unnecessary check in 'setFee()'

- **Location(s):** Vault.sol#169-170
- **Description:** Within `Vault.sol` the function `setFee()` reverts if the input `newFee` is outside the allowed range of values [`_MIN_FEE` ; `_MAX_FEE`] which are constants respectively equal to `0` and `15000`.

```
uint32 private constant _FEE_BASIS = 100_000;
uint32 private constant _MIN_FEE = 0; // 0%
uint32 private constant _MAX_FEE = 15_000; // 15%

...

function setFee(uint32 newFee) external onlyOperator {
```

[5]https://explorer.execution.mainnet.lukso.network/address/0x9F49a95b0c3c9e2A6c77a16C177928294c0F6F04?tab=contract
[6]https://explorer.execution.mainnet.lukso.network/address/0x2Cb02ef26aDDAB15686ed634d70699ab64F195f4?tab=contract
[7]https://explorer.execution.mainnet.lukso.network/address/0xe460f0cb1227399b129715895157e8cb443b269e
[8]https://github.com/Universal-Page/contracts/blob/136c6dbbc8c7941b9d9a12a4867bd9dfe66dfb76/src/pool/Vault.sol
[9]https://github.com/Universal-Page/contracts/blob/136c6dbbc8c7941b9d9a12a4867bd9dfe66dfb76/src/pool/Vault.sol
[10]https://explorer.execution.mainnet.lukso.network/address/0x2Cb02ef26aDDAB15686ed634d70699ab64F195f4?tab=contract

```
        if (newFee > _FEE_BASIS) {
            revert InvalidAmount(newFee);
        }
        if (newFee < _MIN_FEE || newFee > _MAX_FEE) {
            revert InvalidAmount(newFee);
        }
        ...
    }
```

However before the mentioned check, `newFee` is also supposed to be lower than or equal to `_FEE_BASIS` (which is a constant equal to `100000`). Since `_FEE_BASIS` is greater than `_MAX_FEE`, that check is unnecessary as `newFee` should always be lower than `15000` .

- **Recommendation:** Consider removing the following check from `setFee()`

```
        if (newFee > _FEE_BASIS) {
            revert InvalidAmount(newFee);
        }
```

- **Status:** Resolved.
- **Updates:**
  - `[Extropy, 06/01/2025]`: Fixed in commit ba23c56[11]

4.8 | [INFO] ‘feeRecipient‘ name and comment are misleading

- **Location(s):** Vault.sol#94-95
- **Description:** Within `Vault.sol` the state variable `feeRecipient` is defined as the "Recipient of the vault fee"

```
    // Recipient of the vault fee
    address public feeRecipient;
```

However `claimFees()` is not ensuring that `feeRecipient` is the `beneficiary` address (namely who receives the fees) but checks that `feeRecipient` is the `msg.sender` (namely who trigger the claim operation).

```
    function claimFees(uint256 amount, address beneficiary) external override
                                        nonReentrant whenNotPaused {
        if (beneficiary == address(0)) {
            revert InvalidAddress(beneficiary);
        }
        address account = msg.sender;
        if (account != feeRecipient) {
            revert CallerNotFeeRecipient(account);
        }
       ...
    }
```

[11]https://github.com/Stakingverse/pool-contracts/commit/ba23c565ca874ad35888386ba6e32261d886ab5b

- **Recommendation:** Consider changing the name of the variable `feeRecipient` as well as changing the comment at line 94 to reflect the code behavior.
- **Status:** Partially resolved.
- **Updates:**
  - `[StakingVerse, 06/01/2025]`: "For simplicity, the state variable was not renamed to not change the Vault interface and ABI (since `feeRecipient` is defined as `public`, a getter function is automatically created). However, the comment above the state variable was updated to describe better that only this address can call `claimFees(uint256)`."
  - `[Extropy, 06/01/2025]`: Changed comment in commit 2e9f5b5[12]. Variable name not changed instead.

4.9 | [INFO] Redundant modifiers

- **Location(s):** LiquidStakingToken.sol#66, LiquidStakingToken.sol#70
- **Description:** Within `LiquidStakingToken.sol` the functions `pause()` and `unpause()` have respectively `whenNotPaused()` and `whenPaused()` modifiers.

```
function pause() external onlyOwner whenNotPaused {
    _pause();
}

function unpause() external onlyOwner whenPaused {
    _unpause();
}
```

  However this is redundant since the internal functions `_pause()` and `_unpause()` inherited from `PausableUpgreadeable` already have the same modifiers and thus run the same code again.
- **Recommendation:** Consider removing the modifiers `whenNotPaused()` and `whenPaused()` from the functions `pause()` and `unpause()`.
- **Status:** Resolved
- **Updates:**
  - `[Extropy, 06/01/2025]`: Fixed in commit cccc8ce[13]

4.10 | [INFO] 'LST' and 'sLYX' are used interchangeably

- **Location(s):** -
- **Description:** Within the entire codebase in scope the terms `LST Tokens` and `sLYX` are used interchangeably to refer to `sLYX` tokens and this may cause confusion.
- **Recommendation:** Consider refactoring the codebase to use the same naming convention.
- **Status:** Resolved.
- **Updates:**

[12]https://github.com/Stakingverse/pool-contracts/commit/2e9f5b5bef9fdb30acd7bf4284f8dc3133b55080
[13]https://github.com/Stakingverse/pool-contracts/commit/cccc8cedb57261a28e3500d3036d196e2a0cd41b

- [StakingVerse, 06/01/2025]: "The Liquid Staking token contract was renamed to SLYXToken and the term liquidStakingToken was replaced by sLyxToken to use the same naming convention and for consistency across the whole codebase (core source code + tests)."
- [Extropy, 06/01/2025]: Fixed in commit 01e5574[14]

4.11 | [INFO] Centralized functions do not emit events on state change

- **Location(s):** Vault.sol#153 , Vault.sol#210, Vault.sol#218, Vault.sol#448
- **Description:** Within the codebase some functions can be executed only by centralized roles (e.g. owner, operator ...) and they do not emit event when the contract state changes:
  - _setOperator() in Vault.sol
  - allowList() in Vault.sol
  - setRestricted() in Vault.sol
  - registerValidator() in Vault.sol
- **Recommendation:** Consider emitting proper events when a function allows a privileged user to alterate the contract state.
- **Status:** Resolved.
- **Updates:**
  - [StakingVerse, 06/01/2025]: "The recommendation was applied and the new events OperatorChanged, VaultRestrictionChanged, AllowedListChanged and ValidatorRegistered were added and emitted in these functions to reflect when privileged users alternate the state of the Vault contract."
  - [Extropy, 06/01/2025]: Fixed in commit 6ab5cd4[15]

4.12 | [INFO] No upper bound for the loop within 'registerValidators()'

- **Location(s):** Vault.sol#477
- **Description:** Within Vault.sol the function registerValidators() processes each element of the pubKeys array in input with a for loop that iterates from 0 to pubkeys.length.

```
function registerValidators( // @audit-ok
    bytes[] calldata pubkeys,
    bytes[] calldata signatures,
    bytes32[] calldata depositDataRoots
) external {
    uint256 length = pubkeys.length;
    if (length != signatures.length || length != depositDataRoots.length) {
        revert InvalidAmount(length);
    }
    for (uint256 i = 0; i < length; i++) {
        registerValidator(pubkeys[i], signatures[i], depositDataRoots[i]);
    }
}
```

[14] https://github.com/Stakingverse/pool-contracts/commit/01e5574e21fc59b5deaf0423fe92a4e9c2c027b9
[15] https://github.com/Stakingverse/pool-contracts/commit/6ab5cd4e6da27221dd1e4bdfe1cfaf5881554152

However if since there is no maximum value for `pubkeys.length`, the for loop has no upperbound and it can consume all the available gas.

- **Recommendation:** Consider setting a reasonable maximum value for `pubkeys.length`.
- **Status:** Acknowledged.
- **Updates:**
  - `[StakingVerse, 06/01/2025]`: "The team is aware that submitting a very large number of validators can make the function revert. To mitigate this issue, large deposits will be simply submitted in multiple smaller batches to prevent the transaction to revert."

4.13 | [INFO] For loop can revert earlier

- **Location(s):** Vault.sol#448-480
- **Description:** Within `Vault.sol` the function `registerValidators()` takes in an array of `pubkeys` and passes each of them to `registerValidator()` to process them. For each call it checks that `totalUnstaked` is greater than or equal to `DEPOSIT_AMOUNT = 32 ether` to proceed the execution, otherwise it reverts.

```
    if (totalUnstaked < DEPOSIT_AMOUNT) {
        revert InsufficientBalance(totalUnstaked, DEPOSIT_AMOUNT);
```

  However if just one of the calls made within the for loop in `registerValidators()` fails due to insufficient `totalUnstaked` the entire transaction reverts and this may checked before the loop starts iterating.

  Suppose the following scenario:

  - `registerValidators()` is called passing `pubkeys` array of length 6
  - `totalUnstaked` is currently 191e18
  - the first 5 calls to `registerValidator()` succeed while the last one fails as only 31 ethers were accounted into `totalUnstaked` at that point

- **Recommendation:** Consider ensuring within `registerValidators()` that `totalUnstaked` is enough to register all the validators in one call, otherwise revert the execution earlier before the for loop starts.

```
function registerValidators( // @audit-ok
    bytes[] calldata pubkeys,
    bytes[] calldata signatures,
    bytes32[] calldata depositDataRoots
) external {
    uint256 length = pubkeys.length;
    if (length != signatures.length || length != depositDataRoots.length) {
        revert InvalidAmount(length);
    }
    require(totalUnstaked >= length * DEPOSIT_AMOUNT, "totalUnstaked is not
                                                     sufficient to register all the vali

    for (uint256 i = 0; i < length; i++) {
        registerValidator(pubkeys[i], signatures[i], depositDataRoots[i]);
    }
}
```

- **Status:** Acknowledged
- **Updates:**
    - `[Stakingverse, 09/01/2025]`: "Skipped as the check suggested to be added seems redundant. A similar check is performed in the code of the Vault line 462."

4.14 | [INFO] Deposit will be blocked after 'totalValidatorsRegistered' reaches the maximum value allowed

- **Location(s):** Vault.sol#302-305
- **Description:** Within `Vault.sol` the function `deposit()` checks if the amount of LYX being deposited is below the allowed limit in the following way:

```
uint256 newTotalDeposits =
    Math.max(totalValidatorsRegistered * DEPOSIT_AMOUNT, totalStaked +
     totalUnstaked) + amount;
if (newTotalDeposits > depositLimit) {
    revert DepositLimitExceeded(newTotalDeposits, depositLimit);
}
```

It computes the maximum value between `totalValidatorsRegistered * DEPOSIT_AMOUNT` and `totalStaked + totalUnstaked` and adds `amount` to it before comparing it against `depositLimit` state variable.

However `depositLimit` is set within `setDepositLimit()` and in the scenario where the number of registered validators is exactly equal to `_MAX_VALIDATORS_SUPPORTED = 1_000_000` it can be set only equal to:

```
depositLimit = _MAX_VALIDATORS_SUPPORTED * DEPOSIT_AMOUNT
// where totalValidatorsRegistered == _MAX_VALIDATORS_SUPPORTED
```

This means that, even if having 1000000 registered validators is a very remote scenario, it will not be possible to call `deposit()` again. Indeed when the the number of validators reaches the threshold `_MAX_VALIDATORS_SUPPORTED`, the `deposit()` function will always revert at line 305 because `depositLimit` can not be increased anymore and `newTotalDeposits` will be always greater to `depositLimit` because:

- if `Math.max(totalValidatorsRegistered * DEPOSIT_AMOUNT, totalStaked + totalUnstaked)` is the first term , adding to it whatever `amount` will make `newTotalDeposits > depositLimit`
- if `Math.max(totalValidatorsRegistered * DEPOSIT_AMOUNT, totalStaked + totalUnstaked)` is the second term, it implicitly makes `newTotalDeposits > depositLimit`

- **Recommendation:** Consider ensuring that there will be always a way to unlock deposits for users.
- **Status:** Acknowledged.

5 | Test Coverage

5.1 | Solidity tests

```
npm run test


> StakingVerse Liquid Staking Token@0.0.1-alpha-0 test
> forge test

[] Compiling...
No files changed, compilation skipped

Ran 5 tests for test/Deployment.t.sol:Deployment
[PASS] test_cannotInitializeLSTImplementationContract() (gas: 14395)
[PASS] test_contractIsNotPausedOnDeployment() (gas: 18361)
[PASS] test_deploymentParametersOfLSTToken() (gas: 52359)
[PASS] test_shouldHaveSetCorrectLinkedStakingVault() (gas: 20841)
[PASS] test_totalSupplyLSTZeroInitially() (gas: 18260)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 1.90ms (483.67µs CPU
↪  time)

Ran 1 test for test/Reentrancy.t.sol:Reentrancy
[PASS] test_cannotReenterOnVaultStakeReceivedViaTransferStake() (gas: 814397)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.39ms (692.92µs CPU
↪  time)

Ran 3 tests for test/Withdraw.t.sol:Withdraw
[PASS] test_cannotWithdrawIfHavingLST() (gas: 281767)
[PASS] test_cannotWithdrawWithLSTContractAddressAsBeneficiary() (gas: 212612)
[PASS] test_depositConvertTransferBurnWithdraw() (gas: 313810)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 11.90ms (2.46ms CPU
↪  time)

Ran 8 tests for test/Pause.t.sol:Pausable
[PASS] test_cannotBurnLSTokensIfContractIsPaused() (gas: 344053)
[PASS] test_cannotCallPauseAgainWhenContractIsAlreadyPaused() (gas: 52860)
[PASS] test_cannotCallUnpauseIfContractIsAlreadyUnpause() (gas: 25870)
[PASS] test_cannotMintLSTTokensIfContractIsPaused() (gas: 296629)
[PASS] test_onlyContractOwnerCanPause(address) (runs: 10000, : 25077, ~: 25077)
[PASS] test_onlyOwnerCanUnpauseContract(address) (runs: 10000, : 56707, ~: 56707)
[PASS] test_shouldPauseContractWhenOwnerCallPause() (gas: 49328)
[PASS] test_shouldUnpauseContractWhenOwnerCallUnpause() (gas: 36404)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 895.84ms (894.98ms
↪  CPU time)

Ran 6 tests for test/Rewards.t.sol:Rewards
[PASS] test_TwoUsersDepositAndBurnLSTAfterAccruedReward() (gas: 728049)
[PASS] test_depositOnBehalfOfLSTContractAndInflateProportionOfRewards() (gas:
↪  693293)
[PASS] test_getNativeTokenValueShouldAlwaysIncreaseAfterRewards() (gas: 553894)
[PASS] test_roundingErrorRemainsOneWeiEvenWithBigDeposits() (gas: 637697440)
[PASS] test_shouldMintMoreTokensWhenTransferStakeAfterAccumulatedRewards() (gas:
↪  587777)
[PASS] test_shouldRetrieveMoreLSTWhenBurningLSTAndRewardsAccruedStake() (gas:
↪  480569)
```

```
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 5.88s (740.88ms CPU
↪  time)

Ran 13 tests for test/TokenBurn.t.sol:TokenBurn
[PASS] test_MintAllLSTAndReTransferStakePartiallyAfterBurningSomeTokens() (gas:
↪  372341)
[PASS] test_Stake100LYXMint50LYXBurn30LYX() (gas: 390597)
[PASS] test_burnOneSingleToken() (gas: 351982)
[PASS] test_burnVerySmallAmountOfToken1Wei() (gas: 351688)
[PASS] test_canBurnAllLSTAsAnOperator() (gas: 322696)
[PASS] test_cannotBurnIfNoTokensWereMintedForUser(uint256,address) (runs: 10000, :
↪  256841, ~: 256853)
[PASS] test_cannotBurnLSTsIfNotAnOperator() (gas: 295552)
[PASS] test_cannotBurnMoreThanAllowanceAsOperator(uint256,uint256) (runs: 10000, :
↪  421206, ~: 422610)
[PASS] test_cannotBurnZeroLST() (gas: 307985)
[PASS] test_cannotConvertBackLSTToLyxMoreThanTokenBalance(uint256,uint256) (runs:
↪  5000, : 305299, ~: 305299)
[PASS] test_emitRightEventsAfterBurningTokens() (gas: 275531)
[PASS] test_shouldReTransferStakeToUserAfterBurningAllTokens() (gas: 292010)
[PASS] test_shouldRevertWhenBurningZeroToken() (gas: 335674)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 11.29s (18.34s CPU
↪  time)

Ran 8 tests for test/TokenTransfer.t.sol:TokenTransfer
[PASS] test_canTransferFullAmountSuccessfully(uint256) (runs: 10000, : 342559, ~:
↪  342559)
[PASS] test_canTransferTokensAndRecipientCanConvertBackToStakeByBurning(uint256)
↪  (runs: 10000, : 351293, ~: 351293)
[PASS] test_canTransferWithZeroAmountSuccessfully() (gas: 321237)
[PASS] test_cannotTransferMoreTokensThanActualBalance(uint256) (runs: 10000, :
↪  295194, ~: 295194)
[PASS] test_cannotTransferTokensToZeroAddress() (gas: 289993)
[PASS] test_cannotTransferTokensWithLSTContractAsRecipient() (gas: 289631)
[PASS] test_cannotTransferTokensWithVaultAsRecipient(address) (runs: 10000, :
↪  283143, ~: 283143)
[PASS] test_cannotTransferTokensWithVaultImplementationAsRecipient() (gas: 293547)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 13.58s (26.85s CPU
↪  time)

Ran 8 tests for test/DepositExtension.t.sol:DepositExtension
[PASS] test_cannotDepositAndTransferStakeToExtensionAddress() (gas: 244981)
[PASS] test_cannotDepositDirectlyOnLSTContractWithLSTContractAddressAsParameter()
↪  (gas: 537490)
[PASS] test_cannotMintLSTTokensViaExtensionIfAmountIsOverDepositLimit(uint256)
↪  (runs: 10000, : 119838, ~: 119838)
[PASS] test_cannotMintLSTTokensViaExtensionToZeroAddress() (gas: 344850)
[PASS] test_onlyLSTTokenContractCanCallExtension(address) (runs: 10000, : 187142,
↪  ~: 187143)
[PASS] test_shouldAllowDepositDirectlyToLSTContract() (gas: 370556)
[PASS] test_shouldEmitTheRightEvents() (gas: 348317)
[PASS] test_shouldRevertWhenPassingZeroAmountViaExtension() (gas: 181708)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 63.10s (4.99s CPU
↪  time)

Ran 11 tests for test/TokenMint.sol:TokenMint
[PASS] test_cannotMintLSTTokensToZeroAddress() (gas: 184765)
[PASS] test_deposit1WeiAndMintLST() (gas: 323662)
```

```
[PASS] test_depositLYXToVaultThenMintTokens(uint256) (runs: 10000, : 292179, ~:
↪  292179)
[PASS] test_shouldIncreaseLSTSharesInVaultWhenTwoUsersTransferStakeToLSTContract()
↪  (gas: 403660)
[PASS] test_shouldMintAllStakeAsLSTTokensWhenCallingTransferStakeToLSTContract()
↪  (gas: 305658)
[PASS] test_shouldMintLSTWithDataWhenPassingDataToTransferStakeFunction(bytes)
↪  (runs: 10000, : 309609, ~: 308920)
[PASS] test_shouldMintPartialStakeAsLSTTokens(uint256) (runs: 10000, : 326968, ~:
↪  326968)
[PASS] test_shouldNotAllowNonVaultToMint(address) (runs: 10000, : 195614, ~:
↪  195614)
[PASS] test_shouldRegisterLSTTokensInLSP5ReceivedAssetsWhenUniversal
↪  ProfileTransferStakeToLSTContract() (gas: 10448050)
[PASS] test_shouldRevertWhenCallingTransferStakeWithAmountLargerThanStake(uint256)
↪  (runs: 10000, : 207641, ~: 207641)
[PASS] test_shouldRevertWhenPassingZeroAmountToTransferStake() (gas: 179513)
Suite result: ok. 11 passed; 0 failed; 0 skipped; finished in 63.10s (88.94s CPU
↪  time)

Ran 2 tests for test/Invariants.t.sol:Invariants
[PASS] invariant_metadataIsConstant() (runs: 9895, calls: 4947500, reverts:
↪  4680146)
[PASS] testInvariant_mintingAffectsTotalSupplyAndBalance(address,uint256) (runs:
↪  10000, : 310974, ~: 310971)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 328.59s (334.47s CPU
↪  time)

Ran 10 test suites in 328.60s (486.46s CPU time): 65 tests passed, 0 failed, 0
↪  skipped (65 total tests)
```

5.2 | Coverage

```
| File                    | % Lines | % Statements | % Branches    | % Funcs        |
|-------------------------|---------|--------------|---------------|----------------|
| LiquidStakingToken      | 76.92%  |     76.19%   | 83.33% (5/6)  | 63.64% (7/11)  |
| LiquidStakingTokenAuto-
          MintExtension  | 88.89%  |     90.91%   | 50.00% (1/2)  | 100.00% (3/3)  |
```

A | Disclaimers

The audit makes no statements or warranty about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purpose

A.1 | Client Confidentiality

This document contains Client Confidential information and may not be copied without written permission.

A.2 | Proprietary Information

The content of this document should be considered proprietary information. Extropy gives permission to copy this report for the purposes of disseminating information within your organisation or any regulatory agency.

B | Slither results

B.1 | LiquidStakingToken.sol

```
slither src/LiquidStakingToken.sol

INFO:Detectors:
LiquidStakingToken._beforeTokenTransfer(address,address,uint256,bytes)
 ↪  (src/LiquidStakingToken.sol#137-150) has external calls inside a loop: to ==
 ↪  address(this) || to == address(stakingVault) || to ==
 ↪  ITransparentUpgradeableProxy(address(stakingVault)).implementation()
 ↪  (src/LiquidStakingToken.sol#144-146)
LiquidStakingToken.getNativeTokenValue(uint256)
 ↪  (src/LiquidStakingToken.sol#99-110) has external calls inside a loop:
 ↪  totalSLYXMinted = stakingVault.sharesOf(address(this))
 ↪  (src/LiquidStakingToken.sol#103)
LiquidStakingToken.getNativeTokenValue(uint256)
 ↪  (src/LiquidStakingToken.sol#99-110) has external calls inside a loop:
 ↪  lstTokenContractStake = stakingVault.balanceOf(address(this))
 ↪  (src/LiquidStakingToken.sol#105)
LiquidStakingToken._afterTokenTransfer(address,address,uint256,bytes)
 ↪  (src/LiquidStakingToken.sol#157-169) has external calls inside a loop:
 ↪  stakingVault.transferStake(from,lstAmountAsLyxStake,data)
 ↪  (src/LiquidStakingToken.sol#167)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/
 ↪  #calls-inside-a-loop
INFO:Detectors:
Version constraint ^0.8.22 contains known severe issues
 ↪  (https://solidity.readthedocs.io/en/latest/bugs.html)
        - VerbatimInvalidDeduplication.
It is used by:
        - ^0.8.22 (src/LiquidStakingToken.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
 ↪  #incorrect-versions-of-solidity
INFO:Slither:src/LiquidStakingToken.sol analyzed (40 contracts with 94 detectors),
 ↪  5 result(s) found1
```

B.2 | LiquidStakingTokenAutoMintExtension.sol

```
slither src/LiquidStakingTokenAutoMintExtension.sol

INFO:Detectors:
Version constraint ^0.8.22 contains known severe issues
 ↪  (https://solidity.readthedocs.io/en/latest/bugs.html)
        - VerbatimInvalidDeduplication.
It is used by:
        - ^0.8.22 (src/LiquidStakingTokenAutoMintExtension.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
 ↪  #incorrect-versions-of-solidity
INFO:Detectors:
Variable LiquidStakingTokenAutoMintExtension._VAULT_ADDRESS
 ↪  (src/LiquidStakingTokenAutoMintExtension.sol#34) is not in mixedCase
Variable LiquidStakingTokenAutoMintExtension._LIQUID_STAKING_TOKEN
 ↪  (src/LiquidStakingTokenAutoMintExtension.sol#35) is not in mixedCase
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
↪  #conformance-to-solidity-naming-conventions
INFO:Slither:src/LiquidStakingTokenAutoMintExtension.sol analyzed (20 contracts
↪  with 94 detectors), 3 result(s) found
```

B.3 | Vault.sol

```
slither lib/UniversalPage-contracts/contracts/src/pool/Vault.sol

INFO:Slither:lib/UniversalPage-contracts/contracts/src/pool/Vault.sol analyzed (14
↪  contracts with 94 detectors), 0 result(s) found
```

extropy