Scanned Code Report

AUDITAGENT

Code Info

Enterprise Scan

Organization
Stakingverse

Scan ID
3

Repository
pool-contracts

Branch
main

Commit Hash
cd818584dc908ff37a6529052e9dac8719bb9901

Contracts in scope

src/SLYXToken.sol
src/StakingverseVault.sol
src/SLYXErrors.sol

Code Statistics

Findings
21

Contracts Scanned
3

Lines of Code
765

Severity	Number
High	1
Medium	5
Low	12
Info	3
TOTAL	21

#	Severity	Contract	Issue Description	Risk Level	Impact	Resolution
H-1	High	StakingverseVault.sol	First depositor exploit due to flawed share calculation	High	Share price inflation	✅ Fixed. Same finding than in MiloTruck audit. Commented in the source code as warning. This contract is used for an upgrade so it will not apply.
M-1	Medium	StakingverseVault.sol	Single oracle has unilateral power to register validators and rebalance	Medium	Potential manipulation of vault balance	☑️ Acknowledged. Design decision. There are no single oracle that have unilateral power. Oracles are controlled by the admin. If an oracle address is compromised, the operator can register another oracle address and remove the compromised one.
M-2	Medium	StakingverseVault.sol	Insufficient validation in validator registration	Medium	Incorrect validator registrations	⚪ Not Applicable / False Positive. Deposit contract on mainnet perform these checks already. https://explorer.lukso.network/address/0xCAfe0000000000000000000000000000CAfe?tab=contract
M-3	Medium	StakingverseVault.sol	Pending withdrawal rebalancing vulnerability	Medium	Manipulated withdrawal state	☑️ Acknowledged. The oracle of Stakingverse can be trusted.
M-4	Medium	SLYXToken.sol	Reentrancy risk in `_afterTokenTransfer` during burn	Medium	Possible reentrancy attack	✅ Mitigated. The `transferStake( ... )` function in the `StakingverseVault` contract contains a `nonReentrant` modifier, which mitigates this issue.
M-5	Medium	StakingverseVault.sol	Complex rebalance logic vulnerability	Medium	Misallocation of rewards or fees	⚪ Not Applicable / False Positive? Not clear, the issue description seems very broad and not clear to me
L-1	Low	SLYXToken.sol	Use of TransparentUpgradeableProxy assumptions in `_beforeTokenTransfer`	Low	Possible failure on upgrade	✅ Fixed. The `implementation()` function can only be called by the proxy admin, which would lead to this internal call to fail. This check was removed.
L-2	Low	StakingverseVault.sol	Operators can adjust fee and deposit limit without timelock	Low	Centralized control risk	☑️ Acknowledged. Operator is controlled mainly by Stakingverse. Would not be the case that drastic fee changes occur out of the sudden. If fee changes, these are communicated to the communities.
L-3	Low	StakingverseVault.sol	Math precision loss in share calculations	Low	Inaccurate balance calculations	☑️ Acknowledged. We know already and cannot really change much in the logic because it is already deployed and upgrading could incur risks.
L-4	Low	StakingverseVault.sol	Missing zero transfer check in `transferStake()`	Low	Unnecessary gas costs	☑️ Acknowledged.
L-5	Low	StakingverseVault.sol	Possible rounding issues in fee calculation	Low	Loss of fees due to rounding down	☑️ Acknowledged.
L-6	Low	SLYXToken.sol	Unprotected initialization parameters	Low	Risk of invalid owner initialization	⚪ Not Applicable / False Positive. Check for `address(0)` is performed in the inheritance of the contract.
L-7	Low	SLYXToken.sol	Incomplete error handling in LSP1 Universal Receiver	Low	Potential failures not caught	⚪ Not Applicable / False Positive. The description of the issue is not clear.
L-8	Low	StakingverseVault.sol	Missing fee recipient validation in `setFee`	Low	Locked fees without recipient	⚪ Not Applicable / False Positive.
L-9	Low	StakingverseVault.sol	No validation for validator existence in `registerValidator`	Low	Easier for malicious oracle to register fake validators	☑️ Acknowledged. The oracle of Stakingverse can be trusted.
L-10	Low	StakingverseVault.sol	Lack of address validation in `transferStake`	Low	Possible silent failures	☑️ Acknowledged. Intended behaviour.
L-11	Low	StakingverseVault.sol	Zero value asset loss risk	Low	Incorrect share price calculations	☑️ Acknowledged. Cannot be applicable as contract is already live and this is an upgrade.
L-12	Low	SLYXToken.sol	Division-by-zero risk in `onVaultStakeReceived`	Low	Possible denial of service	⚪ Not Applicable / False Positive. The Vault is already live in production. It is very unlikely that `totalAssets()` will return zero.
I-1	Info	StakingverseVault.sol / SLYXToken.sol	Vault logic failure if proxy pattern changes	Info	Risk in future upgrades	☑️ Acknowledged. Upgrade will be carefully reviewed including storage layout and the proxy pattern will not be changed (only storage layout will be used).
I-2	Info	SLYXToken.sol	Missing events for critical state changes	Info	Harder to track contract state	⚪ Not Applicable / False Positive. Events are emitted on the parent contract in the underlying functions.
I-3	Info	StakingverseVault.sol / SLYXToken.sol	Missing events for critical operations	Info	Difficult off-chain monitoring	⚪ Not Applicable / False Positive. Events are emitted on the parent contract in the underlying functions.

Code Summary

The protocol implements a liquid staking solution for LYX tokens, allowing users to stake their LYX while maintaining liquidity through sLYX tokens. The system consists of two main components:

1. StakingverseVault: The core staking management contract that:
   - Handles deposits and withdrawals of LYX tokens
   - Manages validator registration and operations
   - Tracks user shares and balances
   - Handles fee collection and distribution
   - Implements a rebalancing mechanism through oracles
   - Supports both restricted (whitelist) and unrestricted modes

2. SLYXToken: A liquid staking token (sLYX) that:
   - Represents a user's stake in the vault
   - Is minted when users stake LYX
   - Can be burned to reclaim the underlying staked LYX
   - Implements the LSP7 token standard
   - Includes built-in exchange rate calculations

Key features include:
- Dynamic exchange rate between LYX and sLYX based on total stake and rewards
- Rebalancing mechanism to account for validator rewards and penalties
- Flexible withdrawal system supporting both immediate and delayed withdrawals
- Fee management system with configurable rates
- Oracle-based validator registration and stake management
- Support for partial withdrawals and rewards distribution

Main Entry Points and Actors:

Vault Contract:
- deposit(address): Used by users to stake LYX and receive shares
- withdraw(uint256, address): Used by stakers to withdraw their staked LYX
- claim(uint256, address): Used by users to claim their pending withdrawals
- transferStake(address, uint256, bytes): Used by stakers to transfer their staked position

sLYX Token Contract:
- burn(address, uint256, bytes): Used by token holders to burn sLYX and reclaim LYX
- transfer/transferFrom: Used by token holders to transfer sLYX tokens

Actors:

- Stakers: Users who deposit LYX into the protocol
- Token Holders: Users who hold and trade sLYX tokens
- Oracles: Trusted entities that manage validator registration and rebalancing
- Operators: Privileged users who can manage protocol parameters
- Fee Recipients: Addresses that can claim accumulated protocol fees

First depositor exploit in Vault

High Risk

The contract is susceptible to a first depositor exploit due to flawed share calculation when totalShares is 0. While the contract attempts to mitigate this by burning `_MINIMUM_REQUIRED_SHARES` (1e3) from the first deposit, this is insufficient protection.

A malicious first depositor can:
1. Make a large first deposit (e.g. 10 ETH) to receive nearly all initial shares minus 1e3
2. Then deposit a small amount (e.g. 0.0001 ETH)
3. Remove the large initial deposit

This manipulation allows inflating the share price significantly for subsequent depositors.

```
function deposit(address beneficiary) public payable override nonReentrant
whenNotPaused {
    // ...
    uint256 shares = _toShares(amount);
    if (shares == 0) {
        revert InvalidAmount(amount);
    }
    // burn minimum shares of first depositor to prevent share inflation and
dust shares attacks.
    if (totalShares == 0) {
        if (shares < _MINIMUM_REQUIRED_SHARES) {
            revert InvalidAmount(amount);
        }
        _shares[address(0)] = _MINIMUM_REQUIRED_SHARES;
        totalShares += _MINIMUM_REQUIRED_SHARES;
        shares -= _MINIMUM_REQUIRED_SHARES;
    }
    // ...
}
```

The current mitigation of burning 1e3 shares is not enough since the attacker can still receive a vastly disproportionate amount of shares with their initial deposit.

2 of 21 Findings — StakingverseVault.sol

Single oracle has unilateral power to register validators and rebalance

• Medium Risk

The contract relies on a single oracle address for critical operations like registering validators (via the function `registerValidator(...)`) and rebalancing (via the function `rebalance()`). A malicious or compromised oracle could register incorrect validator data or manipulate the vault balance state. Below is a critical snippet:

```
function registerValidator(bytes calldata pubkey, bytes calldata signature,
bytes32 depositDataRoot)
    public
    onlyOracle
    nonReentrant
    whenNotPaused
{
    if (totalUnstaked < DEPOSIT_AMOUNT) {
        revert InsufficientBalance(totalUnstaked, DEPOSIT_AMOUNT);
    }
    if (_registeredKeys[pubkey]) {
        revert ValidatorAlreadyRegistered(pubkey);
    }
    ...
}
```

By checking `if (!isOracle(oracle)) revert CallerNotOracle(oracle);`, the code ensures only one privileged oracle can invoke these steps, but that same privilege grants sweeping control over validator registration and rebalancing logic if misused or compromised.

3 of 21 Findings — StakingverseVault.sol

Insufficient Validation in Validator Registration

• Medium Risk

The `registerValidator()` function does not validate the length of the input parameters (pubkey, signature) which could lead to incorrect validator registrations:\n\n`solidity\nfunction registerValidator(\n bytes calldata pubkey,\n bytes calldata signatu`

Pending withdrawal rebalancing vulnerability

Medium Risk

In the `rebalance()` function, the calculation of completed withdrawals could be manipulated by an oracle since there's no validation of the actual completed withdrawals from validators:

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

A malicious oracle could:
1. Cause withdrawals to be marked as complete before they actually are
2. Front-run legitimate withdrawals
3. Manipulate the timing of rebalancing to favor certain withdrawers

This could lead to some users being unable to withdraw their funds as expected.

Reentrancy Risk in SLYXToken _afterTokenTransfer Hook During Burn

Medium Risk

In the SLYXToken contract the _afterTokenTransfer hook is used during a token burn (when tokens are sent to the zero address) to convert burned sLYX tokens into staked LYX by calling the linked Vault's transferStake function. The implementation does not employ any reentrancy guard, meaning that a malicious or compromised stakingVault contract could potentially reenter SLYXToken during this external call. This reentrancy risk can lead to state inconsistencies or allow an attacker to manipulate the exchange rate between sLYX and staked LYX. An excerpt of the vulnerable code is as follows:

```
function _afterTokenTransfer(address from, address to, uint256 amount, bool,
bytes memory data) internal virtual override {
    if (to == address(0)) {
        uint256 sLyxTokenContractStake =
stakingVault.balanceOf(address(this));
        uint256 totalSLYXMintedBeforeBurning = totalSupply() + amount;
        uint256 amountAsStakedLYX = amount.mulDiv(sLyxTokenContractStake,
totalSLYXMintedBeforeBurning);
        stakingVault.transferStake(from, amountAsStakedLYX, data);
    }
}
```

The absence of a reentrancy guard here means that during the execution of transferStake, control could be transferred back into SLYXToken and potentially trigger unintended recursive calls.

Complex Rebalance Logic Vulnerability in StakingverseVault

Medium Risk

The rebalance function in the StakingverseVault contract is responsible for readjusting the vault's accounting by recalculating staked, unstaked, and claimable funds, and by distributing rewards via fee deductions. The function involves multiple arithmetic operations, conditional adjustments, and state updates, all of which must correctly handle rounding errors and edge cases. A miscalculation or logical flaw in this complex process could result in an incorrect allocation of rewards or fees, potentially locking user funds or misappropriating rewards. A representative code excerpt is shown below:

```
function rebalance() external onlyOracle nonReentrant 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;
    uint256 staked = totalStaked - completedWithdrawal;
    uint256 unstaked = balance - totalFees - totalClaimable -
completedWithdrawal;
    uint256 claimable = totalClaimable + completedWithdrawal;
    // Additional adjustments and fee calculations
    if (unstaked - partialWithdrawal > totalUnstaked) {
         uint256 rewards = unstaked - partialWithdrawal - totalUnstaked;
         uint256 feeAmount = Math.mulDiv(rewards, fee, _FEE_BASIS);
         totalFees += feeAmount;
         unstaked -= feeAmount;
    }
    emit Rebalanced(totalStaked, totalUnstaked, staked, unstaked);
    totalClaimable = claimable;
    totalUnstaked = unstaked;
    totalStaked = staked;
}
```

Due to the intricate dependencies among these values, any miscalculation—whether from rounding errors or from logical flaws—could have severe consequences on the vault's financial integrity.

7 of 21 Findings

SLYXToken.sol

Use of TransparentUpgradeableProxy assumptions in sLYX token

`_beforeTokenTransfer`

• Low Risk

Within `_beforeTokenTransfer(...)`, the code reverts if the recipient is the vault logic implementation address. This relies on the assumption of a Transparent Upgradeable Proxy pattern. If, in the future, the vault uses a different proxy pattern (UUPS, or a newly upgraded implementation), this check may fail or revert. Below is the relevant snippet:

```
try ITransparentProxy(address(stakingVault)).implementation() returns
(address implementation) {
    if (to == implementation) {
        revert InvalidRecipientForSLYXTokensTransfer(to);
    }
} catch {}
```

8 of 21 Findings

StakingverseVault.sol

Operators can adjust fee and deposit limit without timelock

• Low Risk

The operator or owner can call `setFee(...)` and `setDepositLimit(...)` at any point, instantly altering the fee amount or deposit limit. This centralizes control, potentially surprising stakers who deposit before it changes. Shown below:

```
function setFee(uint32 newFee) external onlyOperator {
    if (newFee < _MIN_FEE || newFee > _MAX_FEE) {
        revert InvalidAmount(newFee);
    }
    uint32 previousFee = fee;
    fee = newFee;
    emit FeeChanged(previousFee, newFee);
}
```

No timelock or staged change mechanism is employed, so depositors cannot prepare for potential large fee increases.

9 of 21 Findings — StakingverseVault.sol

Math Precision Loss in Share Calculations

• Low Risk

The contract performs multiple division operations when calculating shares which can lead to significant precision loss. This is particularly concerning in the `_toShares()` and `_toBalance()` functions:\n\n`solidity\nfunction _toBalance(uint256 shares) private view returns (uint256) {\n if (total`
functions are used in critical operations like deposits and withdrawals. The precision loss can accumulate over time and lead to discrepancies between the actual value and calculated shares, potentially affecting user balances.

10 of 21 Findings — StakingverseVault.sol

Missing Zero Transfer Check in transferStake()

• Low Risk

While the `transferStake()` function checks if the amount is zero before proceeding with the transfer, it doesn't check if the recipient (`to` address) is the same as the sender (`msg.sender`). This could lead to unnecessary gas costs and event emissions for self-transfers.\n\n`solidity\nfunction transferStake(address to, uint256 amount, bytes calldata data) {\n if (`

11 of 21 Findings — StakingverseVault.sol

Possible Rounding Issues in Fee Calculation

• Low Risk

The fee calculation in the `rebalance()` function may result in rounding issues due to integer division:\n\n`solidity\nuint256 feeAmount = Math.mulDiv(rewards, fee, _FEE_BASIS);\n`\n\nWith `_FEE_BASIS = 100_000`, small reward amounts could result in zero fees due to rounding down, even when fees should be collected. If rewards are consistently small enough to round to zero, this could lead to a loss of fees for the protocol.

12 of 21 Findings — SLYXToken.sol

Unprotected Initialization Parameters

• Low Risk

The `initialize()` function in `SLYXToken` does not validate the `tokenContractOwner_` parameter:\n\n`solidity\nfunction initialize(address tokenContractOwner_, IVault stakingVault_) external` could lead to the contract being initialized with an invalid owner address.

13 of 21 Findings — SLYXToken.sol

Incomplete Error Handling in LSP1 Universal Receiver

• Low Risk

The contract's handling of LSP1 UniversalReceiver hooks in `onVaultStakeReceived()` does not handle potential failures or reverts from the hook execution:\n\n`solidity\nfunction onVaultStakeReceived(address from, uint256 amount, bytes calldata data`

14 of 21 Findings — StakingverseVault.sol

Missing fee recipient validation in `setFee`

• Low Risk

The `setFee()` function allows setting a non-zero fee without having a fee recipient address set. This could lead to fees being permanently locked in the contract.

```
function setFee(uint32 newFee) external onlyOperator {
    if (newFee < _MIN_FEE || newFee > _MAX_FEE) {
        revert InvalidAmount(newFee);
    }
    uint32 previousFee = fee;
    fee = newFee;
    emit FeeChanged(previousFee, newFee);
}
```

If `setFee` is called to set a non-zero fee before setting a fee recipient via `setFeeRecipient`, any collected fees would be permanently locked as there would be no valid recipient address to claim them.

No validation for validator existence in `registerValidator`

• Low Risk

The `registerValidator` function only checks if the validator public key has been previously registered with the contract, but does not validate that it corresponds to an actual validator in the beacon chain or that the signature is valid for that validator:

```
function registerValidator(bytes calldata pubkey, bytes calldata signature,
bytes32 depositDataRoot)
    public
    onlyOracle
    nonReentrant
    whenNotPaused
{
    if (_registeredKeys[pubkey]) {
        revert ValidatorAlreadyRegistered(pubkey);
    }
    _registeredKeys[pubkey] = true;
    totalValidatorsRegistered += 1;
    ...
}
```

While this relies on oracle integrity, having no validation makes it easier for a compromised or malicious oracle to register non-existent validators.

16 of 21 Findings — StakingverseVault.sol

Lack of address validation in `transferStake`

• Low Risk

The `transferStake` function does not validate that the recipient address supports receiving stakes even though it may make a callback to that address:

```
function transferStake(address to, uint256 amount, bytes calldata data)
external override nonReentrant whenNotPaused {
    // ...
    if (ERC165Checker.supportsInterface(to,
type(IVaultStakeRecipient).interfaceId)) {
        IVaultStakeRecipient(to).onVaultStakeReceived(account, amount, data);
    }
}
```

If the recipient is a contract that does not correctly implement `IVaultStakeRecipient` but returns true for the interface check, the callback could fail silently or behave unexpectedly.

17 of 21 Findings — StakingverseVault.sol

Zero value asset loss risk

• Low Risk

The `totalAssets()` function could return a zero value even when there are actual assets in the contract:

```
function totalAssets() public view override returns (uint256) {
    return totalStaked + totalUnstaked + totalClaimable -
totalPendingWithdrawal;
}
```

If `totalPendingWithdrawal` equals or exceeds the sum of other balances, `totalAssets()` would return 0 or underflow. This could cause issues with share price calculations in `_toShares` and `_toBalance` functions that rely on `totalAssets()`.

Division-by-Zero Risk in onVaultStakeReceived Function

Low Risk

The onVaultStakeReceived function in the SLYXToken contract is called by the stakingVault when new LYX stakes are received, triggering the minting of new sLYX tokens. The function calculates the number of sLYX tokens to mint based on the ratio of total shares to total assets from the stakingVault:

```
function onVaultStakeReceived(address from, uint256 amount, bytes calldata
data) external whenNotPaused {
    if (msg.sender != address(stakingVault)) {
        revert OnlyVaultAllowedToMintSLYX(msg.sender);
    }
    uint256 shares = Math.mulDiv(amount, stakingVault.totalShares(),
stakingVault.totalAssets());
    _mint({to: from, amount: shares, force: true, data: data});
}
```

If the stakingVault's totalAssets() function returns zero—due, for example, to improper initialization or an unforeseen state—the division will fail and cause the transaction to revert. This could lead to a denial-of-service condition where sLYX tokens cannot be minted.

19 of 21 Findings

StakingverseVault.sol SLYXToken.sol

Vault logic can fail if proxy pattern is changed during upgrade

• Info

Both the `SLYXToken` and `StakingverseVault` contracts rely on libraries from OpenZeppelin for upgradeability. If an owner upgrades the vault or token contract to use a different proxy approach or modifies the storage layout, certain checks (like the TUP check in `_beforeTokenTransfer(...)`) or storage alignment might break. For instance, changing the proxy storage layout might cause the code below to revert:

```
try ITransparentProxy(address(stakingVault)).implementation() returns
(address implementation) {
    if (to == implementation) {
        revert InvalidRecipientForSLYXTokensTransfer(to);
    }
} catch {}
```

20 of 21 Findings

SLYXToken.sol

Missing Events for Critical State Changes

• Info

Several critical state changes in the contract are not accompanied by event emissions. For example, the `pause()` and `unpause()` functions do not emit events when the contract's state changes:\n\n`solidity\nfunction pause() external onlyOwner {\n _pause(); // No event emitted\n}\n\nfunct`

Missing events for critical operations

Info

Several critical operations in the contracts do not emit events, making it difficult to track important state changes off-chain. For example:

- Initialization of contracts
- Updates to validator status
- Changes in share balances
- Pausing state changes

These operations should emit events to facilitate proper monitoring and indexing of the protocol state.

Disclaimer

Kindly note, the Audit Agent is currently in beta stage and no guarantee is being given as to the accuracy and/or completeness of any of the outputs the Audit Agent may generate, including without limitation this Report. The results set out in this Report may not be complete nor inclusive of all vulnerabilities. The Audit Agent is provided on an 'as is' basis, without warranties or conditions of any kind, either express or implied, including without limitation as to the outputs of the code scan and the security of any smart contract verified using the Audit Agent.

Blockchain technology remains under development and is subject to unknown risks and flaws. This Report does not indicate the endorsement of any particular project or team, nor guarantee its security. Neither you nor any third party should rely on this Report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset.

To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this Report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.