MiloTruck

Stakingverse

Security Review

January 31, 2025

Contents

1 Introduction

1.1 About MiloTruck

MiloTruck is an independent security researcher and 1/4th of the team at Renascence Labs. Currently, he works as a Lead Security Researcher at Spearbit and Lead Auditor at Trust Security.

For private audits or security consulting, please reach out to him on Twitter *@milotruck*.

1.2 Disclaimer

A smart contract security review **can never prove the complete absence of vulnerabilities**. Security reviews are a time, resource and expertise bound effort to find as many vulnerabilities as possible. However, they cannot guarantee the absolute security of the protocol in any way.

2 Risk Classification

Severity Level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality.
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol's functionality/availability.
- Low - Funds are **not** at risk.

2.2 Likelihood

- High - Highly likely to occur.
- Medium - Might occur under specific conditions.
- Low - Unlikely to occur.

3 Executive Summary

3.1 About Stakingverse

Stakingverse offers liquid staking solutions for both LUKSO and Ethereum. To learn more about Stakingverse, please visit https://stakingverse.io.

3.2 Overview

Project Name	Stakingverse
Project Type	Liquid Staking
Language	Solidity
Repository	pool-contracts
Commit Hash	92e9b3777d96c1135719f6b66dc4da29bbb589bc

3.3 Issues Found

High	1
Medium	2
Low	4
Informational	1

4 Findings

4.1 High Risk

4.1.1 `SLYXToken` inherits an unsafe version of `LSP7DigitalAssetInitAbstract`

Context: LSP7DigitalAssetInitAbstract.sol

Description: `SLYXToken` inherits the latest version of `LSP7DigitalAssetInitAbstract`, which has not been audited yet and contains bugs.

For example, whenever a transfer is performed, the `_afterTokenTransfer()` hook and LSP1 callbacks (i.e. `_notifyTokenSender()` and `_notifyTokenReceiver()`) are called twice - once in `_update()` and a second time in `_transfer()`.

As a result, whenever `SLYXToken.burn()` is called, `_afterTokenTransfer()` will be called twice and double the amount of `StakingverseVault` shares will be transferred to the user.

Recommendation: Consider reverting to a safe version of `LSP7DigitalAssetInitAbstract` which has been previously audited, or wait for the latest version to be audited.

Stakingverse: Fixed in commit f5347b.

Upgraded to new patch release of 0.16.3 of the `@lukso/lsp7-contracts` package. Note that this latest version is still not audited. The Stakingverse team will do a final upgrade of this dependency after it has gone an audit and before launching the sLYX Token contract.

MiloTruck: Verified the LSP7 dependencies have been upgraded to a new version. Note that the new version has not been checked to be bug-free as part of this audit.

4.2 Medium Risk

4.2.1 Incorrect accounting in `StakingverseVault.rebalance()`

Context: StakingverseVault.sol#L321-L329

Description: `StakingverseVault.rebalance()` handles LYX received from the beacon chain as such:

```
uint256 balance = address(this).balance;

// account for completed withdrawals
uint256 pendingWithdrawal = totalPendingWithdrawal - totalClaimable;
uint256 completedWithdrawal = Math.min(
    (balance - totalFees - totalUnstaked - totalClaimable) / DEPOSIT_AMOUNT, // actual completed
↪   withdrawals
    pendingWithdrawal / DEPOSIT_AMOUNT // pending withdrawals
        + (pendingWithdrawal % DEPOSIT_AMOUNT == 0 ? 0 : 1) // partial withdrawals
) * DEPOSIT_AMOUNT;

// adjust staked balance for completed withdrawals
uint256 staked = totalStaked - completedWithdrawal;

// take out any claimable balances from unstaked balance prior to calculating rewards.
uint256 unstaked = balance - totalFees - totalClaimable - completedWithdrawal;

// account for withdrawals to claim later
uint256 claimable = totalClaimable + completedWithdrawal;
```

LYX received can come from three sources:

1. Full withdrawals due to validators exiting, which withdraws 32 LYX to the contract.
2. Partial withdrawals due to rewards, which occurs when validators accumulate a balance greater than 32 LYX.

3. LYX directly transferred to the contract via `selfdestruct`.

However, `rebalance()` has no way of distinguishing where the LYX received comes from, as such, its accounting does not appropriately handle increases in the contract's LYX balance.

Assume the vault has the following state:

- 2 validators are staked on the beacon chain, so `totalStaked = 64e18`
- `totalUnstaked = 1e18`
- `totalClaimable = 0`
- `totalPendingWithdrawal = 32e18`
- `totalFees = 0`
- Therefore, `totalAssets() = 64e18 + 1e18 + 0 - 32e18 = 33e18`

The two scenarios below demonstrate how the accounting of `rebalance()` is incorrect.

(1) **Rewards of less than 32 LYX are received from the beacon chain.** For example, 1 LYX is received:

```
balance = 1e18 + 1e18 = 2e18
pendingWithdrawal = 32e18 - 0 = 32e18
completedWithdrawal = 0

staked = 64e18 - 0 = 64e18
unstaked = 2e18 - 0 - 0 - 0 = 2e18
claimable = 0 + 0 = 0
```

The result is that `totalUnstaked` is increased by the rewards received (1 LYX in this case) while `totalClaimable` remains constant. This is problematic as rewards of less than 32 LYX will never be used to resolve pending withdrawals.

(2) **The amount of LYX received from exited validators exceed the amount required for pending withdrawals.** For example, there are 32 LYX of pending withdrawals and 64 LYX is received from 2 validators exiting:

```
balance = 1e18 + 64e18 = 65e18
pendingWithdrawal = 32e18 - 0 = 32e18
completedWithdrawal = 32e18

staked = 64e18 - 3218 = 32e18
unstaked = 65e18 - 0 - 0 - 32e18 = 33e18
claimable = 0 + 32e18 = 32e18
```

After `rebalance()`, `totalAssets() = 32e18 + 33e18 + 32e18 - 32e18 = 65e18`. The result is:

- `totalAssets()` increased by 32 LYX even though the LYX received was from validators exiting, not rewards.
- `totalStaked` is set to 32 LYX even though there are no validators staked anymore.

The correct result would be for `totalStaked` to be decreased to `0`, which matches the number of validators staked and prevents `totalAssets()` from increasing.

Note that these scenarios are not exhaustive; there may be more cases where the accounting in `rebalance()` is incorrect.

Additionally, since there is no way for `rebalance()` to know when a validator has exited, `totalValidatorsRegistered` is never decreased by the number of exited validators and will always remain outdated.

Recommendation: Rework how the vault handles LYX from rewards and validator exits.

Stakingverse: Acknowledged. This is by design. If someone withdraws, they must wait for an actual withdrawal to happen. Rewards are not withdrawals. Once withdrawals happen, the balance of the vault will increase by 32^N.

MiloTruck: Acknowledged.

4.2.2 Allowlist can be bypassed through `StakingverseVault.transferStake()` or transferring SLYX

Context:

- StakingverseVault.sol#L304-L307

Description: In `StakingverseVault`, when `restricted` is set to `true`, only addresses that are whitelisted should be able to own vault shares. This can be seen in `deposit()`, which checks that the caller is in the allowlist when `restricted = true`:

```
address account = msg.sender;
if (restricted && !isAllowlisted(account)) {
    revert InvalidAddress(account);
}
```

However, with the addition of `transferStake()` and `SLYXToken`, users can now transfer shares/SLYX to addresses that are not in the allowlist. Therefore, it is no longer possible for the protocol admin to restrict which addresses can own `StakingverseVault` shares and interact with the protocol.

Recommendation: In `transferStake()`, check that the `to` address is in the allowlist when `restricted` is set to `true`:

```
+ if (restricted && !isAllowlisted(to)) {
+     revert InvalidAddress(to);
+ }
  address account = msg.sender;
```

Note that this still allows SLYX to be transferred to addresses not in the allowlist, but they will not be able to burn their SLYX for vault shares.

Stakingverse: Fixed in commit 1f1ed8. Additionally, a comment was added above the `burn()` function to mention the potential issue of not being able to burn if the function is set to restricted mode.

MiloTruck: Verified, the recommendation was implemented.

4.3 Low Risk

4.3.1 Malicious staker can DOS pending withdrawals

Context:

- StakingverseVault.sol#L417-L432
- StakingverseVault.sol#L269-L271

Description: Whenever the oracle calls `rebalance()`, LYX received by the vault is used to service pending withdrawals by adding them to `totalClaimable`. The remaining LYX is then added to `totalUnstaked`:

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

// adjust staked balance for completed withdrawals
uint256 staked = totalStaked - completedWithdrawal;

// take out any claimable balances from unstaked balance prior to calculating rewards.
uint256 unstaked = balance - totalFees - totalClaimable - completedWithdrawal;

// account for withdrawals to claim later
uint256 claimable = totalClaimable + completedWithdrawal;
```

Afterwards, users with pending withdrawals can call `claim()` to withdraw their remaining LYX. The LYX withdrawn is subtracted from `totalClaimable`:

```
_pendingWithdrawals[account] -= amount;
totalPendingWithdrawal -= amount;
totalClaimable -= amount;
```

However, a malicious staker can DOS pending withdrawals by intentionally reducing `totalClaimable` and increasing `totalUnstaked`.

Assume the following:

- `totalPendingWithdrawal = 32e18`
- After the oracle calls `rebalance()`, `totalUnstaked = 0` and `totalClaimable = 32e18`.

This means there is 32 LYX worth of pending withdrawals and all of it can be withdrawn using `claim()`.

Before any users call `claim()` to resolve their pending withdrawals, a malicious staker does the following:

- Call `withdraw()` with `amount = 32e18`:
  - Since `totalUnstaked = 0`, `immediateAmount = 0` and `delayedAmount = 32e18`.
  - `totalPendingWithdrawal = 32e18 + 32e18 = 64e18-`
  - He now has 32 LYX of pending withdrawals.
- Call `claim()` with `amount = 32e18`:
  - `totalPendingWithdrawal = 64e18 - 32e18 = 32e18`
  - `totalClaimable = 32e18 - 32e18 = 0`
- Call `deposit()` to re-deposit the 32 LYX withdrawn.

The result is that `totalClaimable` has been reset to `0` and the LYX originally allocated to pending withdrawals is transferred to `totalUnstaked`.

If a malicious staker performs this attack whenever `totalClaimable` is non-zero, pending withdrawals will be DOSed and users with pending withdrawals will have their funds stuck.

Recommendation: Since the vault is already live and fixing this issue would mostly likely require a re-design of how the vault handles withdrawals, consider documenting this issue.

Additionally, constantly monitor the vault to ensure withdrawals are not blocked for an extended period of time.

Stakingverse: Acknowledged. There is no "transfer from claimable to unstaked". Once someone withdraws and stake goes to pending withdrawals, it will only go to claimable once 32^N LYX arrives in the contract due to withdrawals and rebalancing. Users must claim to get LYX into their address. Users must send their LYX from their address to deposit to vault. Only then LYX will go back to unstaked.

MiloTruck: Acknowledged.

4.3.2 Malicious stakers can force withdrawals to become pending through front-running

Context: StakingverseVault.sol#L362-L367

Description: Whenever a user calls `StakingverseVault.withdraw()` to withdraw funds, the amount of LYX to withdraw is separated into `immediateAmount` and `delayedAmount`:

```
uint256 immediateAmount = amount > totalUnstaked ? totalUnstaked : amount;
uint256 delayedAmount = amount - immediateAmount;

totalUnstaked -= immediateAmount;
totalPendingWithdrawal += delayedAmount;
_pendingWithdrawals[beneficiary] += delayedAmount;
```

Any unstaked LYX held by the vault (i.e. `totalUnstaked`) can immediately be withdrawn by the user (i.e. `immediateAmount`), while the remaining withdrawal amount (i.e. `delayedAmount`) can only be withdrawn after more liquidity is unstaked by exiting validators.

However, this allows a malicious staker to force users to have to wait to withdraw their funds by front-running their calls to `withdraw()`. For example:

- Assume that `totalUnstaked = 10e18`.
- Bob calls `withdraw()` with `amount = 10e18`, which should be immediately withdrawable.
- Alice, who is a malicious staker, front-runs Bob to call `withdraw()` with `amount = 10e18` first:
  - Since `totalUnstaked == amount`, `immediateAmount = 10e18` and `delayedAmount = 0`.
  - Alice withdraws 10 LYX.
  - `totalUnstaked` is decreased to `0`.
- When Bob's call to `withdraw()` is executed:
  - Since `totalUnstaked == 0`, `immediateAmount = 0` and `delayedAmount = 10e18`.
  - Bob's withdrawal becomes pending.
- Alice re-deposits the 10 LYX she withdrew.

In the example above, Alice forces Bob's withdrawal to become pending by front-running his call to `withdraw()`. Bob now has to wait for the protocol to exit validators and call `rebalance()` to service his withdrawal.

Recommendation: Since the vault is already live and fixing this issue would mostly likely require a re-design of how the vault handles withdrawals, consider documenting this issue.

Additionally, constantly monitor the vault to ensure withdrawals are not blocked for an extended period of time.

Stakingverse: Acknowledged.

MiloTruck: Acknowledged.

4.3.3 `SLYXToken.onVaultStakeReceived()` incorrectly calculates shares on first deposit

Context:

- SLYXToken.sol#L94
- StakingverseVault.sol#L321-L329

Description: In `SLYXToken.onVaultStakeReceived()`, the amount of SLYX minted to the user is calculated by converting `amount` to shares with the vault's `totalShares` and `totalAssets`:

```
uint256 shares = Math.mulDiv(amount, stakingVault.totalShares(), stakingVault.totalAssets());
```

This matches the number of shares minted to the `SLYXToken` contract in `StakingverseVault.deposit()`.

However, when `StakingverseVault.deposit()` is called for the first time, `_MINIMUM_REQUIRED_SHARES` is subtracted from the number of shares minted to the `beneficiary`:

```
// 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;
}
```

Therefore, if the vault's first deposit is performed with `SLYXToken` (i.e. calling `StakingverseVault.deposit()` with `beneficiary` as the `SLYXToken` contract when there are no previous deposits), the share calculation in `SLYXToken.onVaultStakeReceived()` would be incorrect as it does not subtract `_MINIMUM_REQUIRED_SHARES`. This ends up minting an additional `1e3` SLYX to the user.

Note that this is currently not a risk as `SLYXToken` is meant to be integrated with a vault that is already live.

Recommendation: If `SLYXToken` is ever used with a newly deployed `StakingverseVault` in the future, after deployment, consider performing a first deposit with `beneficiary` not set to `SLYXToken`.

Stakingverse: Fixed in commit 8387e2. This cannot occur as vault is currently already live and has funds. However, we have added a comment regarding this issue for anyone reusing the codebase.

MiloTruck: Verified.

4.3.4 Issues with the `nonReentrant` modifier on `StakingverseVault` functions

Context:

- StakingverseVault.sol#L300
- StakingverseVault.sol#L414
- StakingverseVault.sol#L503-L508

Description: `deposit()` and `rebalance()` are missing the `nonReentrant` modifier:

```
function deposit(address beneficiary) public payable override whenNotPaused {
```

```
function rebalance() external onlyOracle whenNotPaused {
```

Additionally, `transferStake()` specifies the `whenNotPaused` modifier before `nonReentrant`:

```
function transferStake(address to, uint256 amount, bytes calldata data)
    external
    override
    whenNotPaused
    nonReentrant
{
```

However, it is a good practice to specify `nonReentrant` as the first modifier in protect against other modifiers having an external call.

Recommendation: Add `nonReentrant` to `deposit()` and `rebalance()`. Additionally, modify `transferStake()` to specify `nonReentrant` first.

Stakingverse: Fixed in commit b70e65.

MiloTruck: Verified, the recommendation was implemented.

4.4 Informational

4.4.1 Minor code improvements

Context: SLYXToken.sol#L34-L40

Description/Recommendation:

`SLYXToken` does not need to inherit `LSP7DigitalAssetInitAbstract` as it is already inherited by `LSP7BurnableInitAbstract`:

```
  contract SLYXToken is
      IVaultStakeRecipient,
      ISLYX,
-     LSP7DigitalAssetInitAbstract,
      LSP7BurnableInitAbstract,
      PausableUpgradeable
  {
```

Stakingverse: Fixed in commit 446985.

MiloTruck: Verified, the recommendation was implemented.