RENASCENCE

Morpheus Lumerin Node Audit Report

Version 2.0

Audited by:

SpicyMeatball

bytes032

January 3, 2025

Contents

1 Introduction

1.1 About Renascence

Renascence Labs was established by a team of experts including HollaDieWaldfee, MiloTruck, alexxander and bytes032.

Our founders have a distinguished history of achieving top honors in competitive audit contests, enhancing the security of leading protocols such as Reserve Protocol, Arbitrum, MaiaDAO, Chainlink, Dodo, Lens Protocol, Wenwin, PartyDAO, Lukso, Perennial Finance, Mute and Taurus.

We strive to deliver tailored solutions by thoroughly understanding each client's unique challenges and requirements. Our approach goes beyond addressing immediate security concerns; we are dedicated to fostering the enduring success and growth of our partners.

More of our work can be found here.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an 'as-is' and 'as-available' basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

1.3.1 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
- Low - Funds are **not** at risk

1.3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

2 Executive Summary

2.1 About Morpheus Lumerin Node

The facet contracts under review from the Morpheus Lumerin Node project — ModelRegistry.sol, ProviderRegistry.sol, Marketplace.sol, and SessionRouter.sol — facilitate a marketplace be- tween users and computing resource providers.

Providers and models are registered through the Registry contracts and are required to lock up a stake at registration. Providers participate in the marketplace by offering a bid consisting of a pri- cePerSecond for a cho- sen model, indicating the price of their compute efforts. A user will choose a provider, the provider will generate a signature, and the user will initiate a session with the provider, locking up a MOR token stake.

The amount of the stake determines the length of the sessions. During the session, the provider earns MOR tokens from the MorpheusAIs Compute pool. At the end of a session, the provider re- ceives the earned MOR reward, and the user receives their staked funds back.

2.2 Overview

Project	Morpheus Lumerin Node
Repository	Morpheus-Lumerin-Node
Commit Hash	79558d805661...
Date	25 December 2024 - 30 December 2024

2.3 Issues Found

Severity	**Count**
High Risk	1
Medium Risk	3
Low Risk	0
Informational	0
Total Issues	**4**

3 Findings Summary

ID	Description	Status
H-1	Stakers' funds may become stuck in the Diamond contract	Resolved
M-1	ProvidersDelegate owner can bypass minDeregistrationTimeout constraint	Acknowledged
M-2	ProvidersDelegate cannot claim the withheld reward from the SessionRouter	Resolved
M-3	Paying bid fee can break accounting in ProvidersDelegate	Resolved

4 Findings

High Risk

[H-1] Stakers funds may become stuck in the Diamond contract

Context: ProviderRegistry.sol#L77-L87 ProviderRegistry.sol#L99-L105

Description: The `ProviderRegistry` contract implements a mechanism that limits the amount of stake a provider can withdraw after deregistration:

```
    function providerDeregister(address provider_) external {
        ---SNIP---
»       uint256 withdrawAmount_ = _getWithdrawAmount(provider);

        provider.stake -= withdrawAmount_;
        provider.isDeleted = true;

        providersStorage.activeProviders.remove(provider_);

        if (withdrawAmount_ > 0) {
            BidsStorage storage bidsStorage = _getBidsStorage();
»           IERC20(bidsStorage.token).safeTransfer(provider_, withdrawAmount_);
        }

        emit ProviderDeregistered(provider_);
    }
```

This limit is periodically updated based on the rewards claimed by the provider in the `SessionRouter` contract:

```
    function _claimForProvider(Session storage session, uint256 amount_) private {
        Bid storage bid = _getBidsStorage().bids[session.bidId];
        Provider storage provider = _getProvidersStorage().providers[bid.provider];

        if (block.timestamp > provider.limitPeriodEnd) {
»           provider.limitPeriodEnd = uint128(block.timestamp) +
PROVIDER_REWARD_LIMITER_PERIOD;
            provider.limitPeriodEarned = 0;
        }

        uint256 providerClaimLimit_ = provider.stake - provider.limitPeriodEarned;

        amount_ = amount_.min(providerClaimLimit_);
        if (amount_ == 0) {
            return;
        }

        session.providerWithdrawnAmount += amount_;
»       provider.limitPeriodEarned += amount_;
```

For example, if a provider has staked 1000 MOR and claimed 800 MOR in rewards, they can unstake only 200 MOR during the current period:

```
    function _getWithdrawAmount(Provider storage provider) private view returns
    (uint256) {
        if (block.timestamp > provider.limitPeriodEnd) {
            return provider.stake;
        }

»      return provider.stake - provider.limitPeriodEarned;
    }
```

The `ProvidersDelegate` contract acts as a provider staking on behalf of its stakers. However, it can call `providerDeregister` to unstake users' funds only once. If this occurs during a period in which rewards were claimed, users' funds may remain stuck in the `Diamond` contract with no way to retrieve them.

Recommendation: To address this issue, the delegate should ensure that `providerDeregister` fully transfers the `provider.stake` value. This can be achieved by verifying that `provider.limitPeriodEarned` is zero or `block.timestamp > provider.limitPeriodEnd` before calling the function.

Client:

Renascence:

Medium Risk

[M-1] ProvidersDelegate owner can bypass minDeregistrationTimeout constraint

Context: DelegateFactory.sol#L62 ProvidersDelegate.sol#L219-L222

Description: When the `ProvidersDelegate` contract is deployed, the factory enforces a check on the `deregistrationOpensAt_` parameter provided by the caller:

```
    function deployProxy(
        address feeTreasury_,
        uint256 fee_,
        string memory name_,
        string memory endpoint_,
        uint128 deregistrationOpensAt_
    ) external whenNotPaused returns (address) {
»       if (deregistrationOpensAt_ <= block.timestamp + minDeregistrationTimeout) {
            revert InvalidDeregistrationOpenAt(deregistrationOpensAt_,
            uint128(block.timestamp + minDeregistrationTimeout));
        }
```

This ensures that the deregistration cannot open earlier than the specified `minDeregistrationTimeout`. However, once the contract is deployed, the owner can bypass this restriction and deregister a provider at any time without waiting for the deregistration checkpoint:

```
    function providerDeregister(bytes32[] calldata bidIds_) external {
        if (!isDeregisterAvailable()) {
»           _checkOwner();
        }
```

Recommendation: Modify the contract logic to require that the `factory.minDeregistrationTimeout` period has passed before the owner is allowed to deregister providers.

Client: `minDeregistrationTimeout` is needed to limit users. The contract owner can deregister at any time.

Renascence: Acknowledged.

[M-2] ProvidersDelegate cannot claim the withheld reward from the SessionRouter

Context: SessionRouter.sol#L285-L290

Description: If a session is closed both with a dispute and prematurely, a portion of the provider's reward is withheld until the next day:

```
    function _rewardProviderAfterClose(bool noDispute_, Session storage session, Bid
    storage bid) internal {
        ---SNIP---
        if (!noDispute_ && !isClosingLate_) {
            providerOnHoldAmount = _getProviderOnHoldAmount(session, bid);
        }
»       providerAmountToWithdraw_ -= providerOnHoldAmount;

        _claimForProvider(session, providerAmountToWithdraw_);
    }
```

To receive the remainder of the reward, the provider must manually call the `claimForProvider` function. Unfortunately, the `ProvidersDelegate` contract, which acts as the provider, does not have the ability to call this function. Consequently, some rewards remain undistributed.

Recommendation: Consider adding a function in `ProvidersDelegate` to enable calling the `claimForProvider` function in the `SessionRouter`.

Client: Fixed in https://github.com/MorpheusAIs/Morpheus-Lumerin-Node/pull/40/commits/62bdf68982f54ce273571520a88c4805d36adf39

Renascence: Fixed as recommended.

[M-3] Paying bid fee can break accounting in ProvidersDelegate

Context: ProvidersDelegate.sol#L234-L240 Marketplace.sol#L86

Description: When the `ProviderDelegate` owner invokes the `postModelBid` function, the contract uses its own reserves to pay the marketplace bid fee:

```
    function postModelBid(
        address provider_,
        bytes32 modelId_,
        uint256 pricePerSecond_
    ) external returns (bytes32 bidId) {
    ---SNIP---

»      IERC20(bidsStorage.token).safeTransferFrom(provider_, address(this),
marketStorage.bidFee);
```

However, these reserves may already be accounted for and included in the current `lastContractBalance`:

```
    function getCurrentRate() public view returns (uint256, uint256) {
        uint256 contractBalance_ = IERC20(token).balanceOf(address(this));

        if (totalStaked == 0) {
            return (totalRate, contractBalance_);
        }

»      uint256 reward_ = contractBalance_ - lastContractBalance;
        uint256 rate_ = totalRate + (reward_ * PRECISION) / totalStaked;
```

Withdrawing funds from the contract can lead to a situation where users are unable to interact with the contract due to an underflow in `getCurrentRate`.

Recommendation: It is advisable to charge the caller of the `postModelBid` function for the marketplace bid fee instead of deducting it from the `ProvidersDelegate`'s reserves.

Client: Fixed in https://github.com/MorpheusAIs/Morpheus-Lumerin-Node/pull/40/commits/b18601f73721b78ed7e52b9177f018a9c038807b

Renascence: Fixed as recommended.