RENASCENCE

Morpheus Lumerin Node Audit Report

Version 2.0

Audited by:

SpicyMeatball

bytes032

November 15, 2024

Contents

1 Introduction **2**
1.1 About Renascence 2
1.2 Disclaimer 2
1.3 Risk Classification 2

2 Executive Summary **3**
2.1 About Morpheus Lumerin Node 3
2.2 Overview 3
2.3 Issues Found 3

3 Findings Summary **4**

4 Findings **5**

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	a53d569c423f...
Date	31 October 2024 - 5 November 2024

2.3 Issues Found

Severity	Count
High Risk	2
Medium Risk	1
Low Risk	2
Informational	3
Total Issues	**8**

3 Findings Summary

ID	Description	Status
H-1	Provider can bypass reward holdback	Resolved
H-2	Funding account tokens are accessible to anyone	Acknowledged
M-1	Anyone can prevent users from registering new model	Resolved
L-1	It is possible to create bids for unregistered models and providers	Resolved
L-2	Users can dispute a valid provider	Acknowledged
I-1	Rogue provider can manipulate model statistics	Acknowledged
I-2	Model fee is not used in the code	Acknowledged
I-3	Stats storage slot is missing ”diamond keyword”	Resolved

4 Findings

High Risk

[H-1] Provider can bypass reward holdback

Context: SessionRouter.sol#L267-L271

Description: When a session closes early and the user flags a dispute, the protocol withholds a portion of the provider's reward:

```
    function _rewardProviderAfterClose(bool noDispute_, Session storage session, Bid
    storage bid) internal {
        uint128 startOfToday_ = startOfTheDay(uint128(block.timestamp));
        bool isClosingLate_ = uint128(block.timestamp) > session.endsAt;

        uint256 providerAmountToWithdraw_ = _getProviderRewards(session, bid, true);
        uint256 providerOnHoldAmount = 0;
»      if (!noDispute_ && !isClosingLate_) {
            providerOnHoldAmount =
                (session.endsAt.min(session.closedAt) -
                startOfToday_.max(session.openedAt)) *
                bid.pricePerSecond;
        }
        providerAmountToWithdraw_ -= providerOnHoldAmount;
        _claimForProvider(session, providerAmountToWithdraw_);
```

However, this withholding approach is ineffective because the provider can immediately claim the remaining amount through the `claimForProvider` function.

Recommendation: Consider implementing a reward-locking mechanism similar to the one used for the user's stake when a session ends prematurely, preventing immediate claim of the withheld rewards.

[H-2] Funding account tokens are accessible to anyone

Context:

- SessionRouter.sol#L104
- SessionRouter.sol#L372-L373

Description: When a user opens a session, they can specify a method of paying the provider for services:

- `isDirectPaymentFromUser = true` means the user pays the provider from their stake.
- `isDirectPaymentFromUser = false` means the provider is paid from the fundingAccount specified during SessionRouter creation.

However, there are no specific conditions to choose the latter option, allowing users to access protocol services for free since the payment comes from the funding account.

Recommendation: Consider restricting access to the funding account to selected providers or users only.

Client: For the present moment this approach has been accepted. That is, when opening a session, the user can choose where the payment to the provider will come from, either from his stake (direct MOR payment) or from the compute pool.

Renascence: Acknowledged.

Medium Risk

[M-1] Anyone can prevent users from registering new model

Context:

- ModelRegistry.sol

Description: The `modelId` passed should be unique for a model to be created. However, given the nature of the blockchain this can mean anyone can prevent users from creating new models.

Assume the following scenario:

1. User A calls createModel(id = 5)
2. User B monitors the mempool and front runs User A's call with the same model id
3. User A's call reverts.

Yes, User B gains nothing from it, but they can essentially create a DOS model for any protocol user and the process can be executed repeatedly with low costs.

Recommendation:

```
+ modelId_ = keccak256(modelId_, msg.sender)
```

Low Risk

[L-1] It is possible to create bids for unregistered models and providers

Context: Marketplace.sol#L67-L72 ModelStorage.sol#L41 ProviderStorage.sol#L39

Description: In the `Marketplace.sol` contract, when a provider attempts to create a bid, the contract performs validation to check if the model and provider are active and registered through their respective registries:

```
    function postModelBid(bytes32 modelId_, uint256 pricePerSecond_) external returns
    (bytes32 bidId) {
        address provider_ = _msgSender();

»      if (!getIsProviderActive(provider_)) {
            revert MarketplaceProviderNotFound();
        }
»      if (!getIsModelActive(modelId_)) {
            revert MarketplaceModelNotFound();
        }
```

However, within the `getIsProviderActive` and `getIsModelActive` functions, entities that are not initialized are also viewed as "active" because the default value of `isDeleted` is `false`:

```
function getIsModelActive(bytes32 modelId_) public view returns (bool) {
    return !_getModelsStorage().models[modelId_].isDeleted;
}

function getIsProviderActive(address provider_) public view returns (bool) {
    return !_getProvidersStorage().providers[provider_].isDeleted;
}
```

As a result, it's possible for bids to be created for non-existent models or by unregistered providers, which undermines the registration checks.

Recommendation:

```
    function getIsModelActive(bytes32 modelId_) public view returns (bool) {
+       return (!_getModelsStorage().models[modelId_].isDeleted &&
_getModelsStorage().models[modelId_].createdAt !=0);
    }

    function getIsProviderActive(address provider_) public view returns (bool) {
+       return (!_getProvidersStorage().providers[provider_].isDeleted &&
!_getProvidersStorage().providers[provider_].createdAt !=0);
    }
```

Client: Resolved in the following commit ec2f394a4b8bf835fb999ea85b7881aa352497e8

Renascence: Fixed as recommended.

[L-2] Users can dispute a valid provider

Context: SessionRouter.sol#L221 SessionRouter.sol#L267-L271

Description: In the `closeSession` function, if the receipt provided by the user does not match the provider's signature, the `noDispute` flag is set to `false`:

```
    function closeSession(bytes calldata receiptEncoded_, bytes calldata signature_)
    external {
        (bytes32 sessionId_, uint32 tpsScaled1000_, uint32 ttftMs_) =
        _extractProviderReceipt(receiptEncoded_);
        ---SNIP---
»       bool noDispute_ = _isValidProviderReceipt(bid.provider, receiptEncoded_,
signature_);
```

This has several effects:

- The `tpsScaled1000_` and `ttftMs_` values from the receipt will not be included in the statistics.
- The provider will incur a penalty, where a portion of their reward will be held back.

Since users receive their stake regardless, they face no repercussions for submitting a mismatched receipt, allowing them to potentially submit incorrect data and cause penalties for providers.

Recommendation: It may be possible to separate reward claims for users and providers. Consider this flow:

- the user calls `closeSession` with a `sessionId` argument and receives their remaining stake while the provider's calculated reward is stored.
- To claim the reward, the provider submits a receipt with `tpsScaled1000_` and `ttftMs_` values, and the statistics are updated accordingly.

Client: That's a good idea. But then the provider would need to call this function every time. I don't know how much it will fit into the overall implementation. Both user and provider will get penalties in case of early closure with a dispute. In turn, if the closing is later, in case of a dispute, we do not record statistics, there are no penalties within the contract. Without changes.

Renascence: Acknowledged.

Informational

[I-1] Rogue provider can manipulate model statistics

Context: SessionRouter.sol#L207 SessionRouter.sol#L309-L329

Description: When a session closes, the `SessionRouter` contract updates statistics for both the model and the provider, using values sent by the provider (`ttftMs_` and `tpsScaled1000_`):

```
function _setStats(
    bool noDispute_,
    uint32 ttftMs_,
    uint32 tpsScaled1000_,
    Session storage session,
    Bid storage bid
) internal {
    ProviderModelStats storage prStats = _providerModelStats(bid.modelId,
    bid.provider);
    ModelStats storage modelStats = _modelStats(bid.modelId);

    prStats.totalCount++;

    if (noDispute_) {
        if (prStats.successCount > 0) {
            // Stats for this provider-model pair already contribute to average model
            stats
            modelStats.tpsScaled1000.remove(int32(prStats.tpsScaled1000.mean),
            int32(modelStats.count - 1));
            modelStats.ttftMs.remove(int32(prStats.ttftMs.mean),
            int32(modelStats.count - 1));
        } else {
            // Stats for this provider-model pair do not contribute
            modelStats.count++;
        }

        // Update provider model stats
        prStats.successCount++;
        prStats.totalDuration += uint32(session.closedAt - session.openedAt);
        prStats.tpsScaled1000.add(int32(tpsScaled1000_), int32(prStats.successCount));
        prStats.ttftMs.add(int32(ttftMs_), int32(prStats.successCount));

        // Update model stats
        modelStats.totalDuration.add(int32(prStats.totalDuration),
        int32(modelStats.count));
        modelStats.tpsScaled1000.add(int32(prStats.tpsScaled1000.mean),
        int32(modelStats.count));
        modelStats.ttftMs.add(int32(prStats.ttftMs.mean), int32(modelStats.count));
    } else {
```

The values `ttftMs_` and `tpsScaled1000_` are taken from a signed receipt:

```
function closeSession(bytes calldata receiptEncoded_, bytes calldata signature_)
external {
    (bytes32 sessionId_, uint32 tpsScaled1000_, uint32 ttftMs_) =
    _extractProviderReceipt(receiptEncoded_);
```

This presents a problem as these values are unverified, meaning the protocol has to fully trust the provider to supply accurate data. Since anyone can register as a provider, malicious actors could intentionally provide misleading data to distort model statistics.

Recommendation: The impact of this issue is challenging to determine without knowing how the stored statistics are used. If these statistics are critical to protocol functionality, consider implementing a provider whitelist or a mechanism to verify `ttftMs_` and `tpsScaled1000_`.

Client: Yes, we're aware of that. There are plans to pass this data using the backend, but it will not protect much against attackers. Perhaps a validation layer will be added in the future. Without changes.

Renascence: Acknowledged.

[I-2] Model fee is not used in the code

Context: ModelRegistry.sol#L30

Description: When registering a model, the caller provides a `fee` value, which is then stored in the `Model` structure. However, this fee value is not referenced anywhere else in the code and appears unused.

Recommendation: Consider removing `fee` field if itsn't used.

Client: Yes, it was in the code, I think this functionality will be extended later, that's why they left it in. Without changes.

Renascence: Acknowledged.

[I-3] Stats storage slot is missing "diamond keyword"

Context:

- StatsStorage.sol

Description:

```
-bytes32 public constant STATS_STORAGE_SLOT = keccak256("diamond.stats.storage");
+bytes32 public constant STATS_STORAGE_SLOT =
keccak256("diamond.standard.stats.storage");
```