Morpheus Lumerin Node Audit Report

Version 2.0

Audited by:

xiaoming90

alexxander

August 8, 2024

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 between 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 `pricePerSecond` 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 receives 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	2a58e01bf039...
Mitigation Hash	e3aef8a6db92...
Date	21 June 2024 - 24 June 2024

2.3 Issues Found

Severity	**Count**
High Risk	8
Medium Risk	7
Low Risk	4
Informational	4
Total Issues	**23**

3 Findings Summary

ID	Description	Status
H-1	Create bid feature will be broken if users manually delete their bids	Resolved
H-2	Staked funds might be lost if multiple sessions is opened within a single block	Resolved
H-3	Total amount of MOR tokens claimed was not updated during session closure	Resolved
H-4	Incorrect compute pool balance	Resolved
H-5	Providers receive fewer MOR tokens than expected during session closure	Acknowledged
H-6	Issues due to the "rounding-down" characteristic of `startOfTheDay` function	Resolved
H-7	There is currently no mechanism to prevent malicious Providers from earning MOR tokens at a zero cost	Acknowledged
H-8	Unexpected revert can occur in `SessionRouter.closeSession()`.	Resolved
M-1	Session's signatures are vulnerable to cross-chain replay attacks	Resolved
M-2	Malicious user can open a session without provider's approval by front-running	Resolved
M-3	Provider can DOS users by closing the session prematurely	Resolved
M-4	Staked MOR tokens will be locked for longer than expected	Resolved
M-5	Re-registering of a model leads to corrupt staked funds accounting	Resolved
M-6	Re-registering of a provider leads to corrupt staked funds accounting.	Resolved
M-7	Providers and Models can participate in Sessions without being registered and staking MOR tokens	Resolved
L-1	ERC-1271 is not supported	Acknowledged
L-2	Re-register of a model will flag the model as `isDeleted: false`, however, the model will not be active	Resolved
L-3	Re-registering a provider will flag the provider as `isDeleted: false`, even though the provider is not active	Resolved
L-4	Takeover of a de-registered model is possible under the assumption that `s.modelMinStake = 0`	Resolved
I-1	Lack of timelock allowing protocol owner to pull assets from user's wallet	Acknowledged
I-2	Lack of validation for delete history	Resolved
I-3	Out of gas error due to unbounded loop	Resolved

ID	Description	Status
I-4	Code improvements	Resolved

4 Findings

High Risk

[H-1] Create bid feature will be broken if users manually delete their bids

Context:

- Marketplace.sol#L129

Description: It was found that the create bid feature will be broken if users call the `deleteModelAgentBid` function to manually delete their bids.

Assume the following scenario:

- At T1, Bob (provider) calls the `postModelAgentBid` function to create a bid with a Bid ID of Bid_1. $Bid_1.createdAt = T1$ and $Bid_1.deletedAt = 0$
- At T2, Bob calls `deleteModelAgentBid` function to delete Bid_1. The following condition within the `deleteModelAgentBid` function will be evaluated as `False`, and Bid_1 will be removed. $Bid_1.deletedAt = T2$

```
bid.createdAt == 0 || bid.deletedAt != 0
T1 == 0 || 0 != 0
False || False
False
```

- At T3, Bob wants to create a new bid again. He calls the `postModelAgentBid` function. Since there is an existing bid, the old bid (Bid_1) will be removed first via the `deleteModelAgentBid` function. When the `deleteModelAgentBid` function is executed, it will revert because the following condition will evaluate as `True`.

```
bid.createdAt == 0 || bid.deletedAt != 0
T1 == 0 || T2 != 0
False || True
True
```

- Bob could no longer create a bid as the `postModelAgentBid` function will always revert. The create bid function is effectively broken for Bob.

Recommendation:

1. Only delete the bid if it has not been deleted yet; OR
2. Return `false` boolean instead of reverting if a bid was not found or has already been deleted

Morpheus: Fixed in Marketplace.sol#L160.

Renascence: The recommendation has been implemented.

[H-2] Staked funds might be lost if multiple sessions is opened within a single block

Context:

- SessionRouter.sol#L140

Description: The session ID of the opened session is constructed by hashing the `abi.encodePacked` of four (4) parameters (`sender, bid.provider, _stake, block.number`), as shown in Line 140 below:

```
File: SessionRouter.sol
107:   function openSession(
108:     uint256 _stake,
109:     bytes memory providerApproval,
110:     bytes memory signature
111:   ) external returns (bytes32 sessionId) {
112:     address sender = msg.sender;
113:
114:     // reverts without specific error if cannot decode abi
115:     (bytes32 bidId, uint128 timestampMs) = abi.decode(providerApproval,
(bytes32, uint128));
..SNIP..
121:     Bid memory bid = s.bidMap[bidId];
..SNIP..
140:     sessionId = keccak256(abi.encodePacked(sender, bid.provider, _stake,
block.number));
141:     s.sessions.push(
142:       Session({
143:         id: sessionId,
144:         user: sender,
..SNIP..
```

In the current implementation, if a user opens two (2) sessions within a single block, the first session will be overwritten by the second one. As a result, the funds staked in the first session will be lost.

Assume that Alice (provider) created two bids (Bid_1 and Bid_2). Bob decided to open sessions with Bid_1 and Bid_2, and got the approval (also signature) from Alice.

Bob submits the two open session transactions with the same stake amount. These two transactions get executed in the same block (Blk-999)

The uniqueness of the Session ID generated depends on `sender, bid.provider, _stake, block.number`. In this case, they are the same for both transactions (`Bob, Alice, 100, Blk-999`), so the Session ID will be the same.

Recommendation: A `nonce` should be included in the generation of `sessionId`. `nonce` should be incremented every time after a session is opened to avoid any collision.

Morpheus: Fixed in SessionRouter.sol#L151.

Renascence: The recommendation has been implemented.

[H-3] Total amount of MOR tokens claimed was not updated during session closure

Context:

- SessionRouter.sol#L226

Description: The `s.totalClaimed` variable keep track of the total amount of MOR claimed by providers. This variable is critical to the protocol's accounting as it is used within the `SessionRouter.getComputeBalance` and `SessionRouter.totalMORSupply` functions.

```
File: AppStorage.sol
114:    uint256 totalClaimed; // total amount of MOR claimend by providers
```

Whenever the providers claim MOR tokens, the `s.totalClaimed` variable must be incremented by the number of MOR tokens claimed. However, it was found that providers claim MOR token within the `SessionRouter.closeSession` function, the `s.totalClaimed` variable was not incremented accordingly, leading the protocol's accounting to be inaccurate.

```
File: SessionRouter.sol
175:    function closeSession(bytes memory receiptEncoded, bytes memory signature)
external {
..SNIP..
226:      session.providerWithdrawnAmount += providerWithdraw;
..SNIP..
246:      // withdraw provider and user funds
247:      s.token.transferFrom(s.fundingAccount, session.provider, providerWithdraw);
..SNIP..
```

As a result, a wide range of issues could occur, such as inflated or deflated value being returned from the `SessionRouter.getComputeBalance` and `SessionRouter.totalMORSupply` functions, leading to users receiving more or less stipend than expected.

Recommendation: The claimed/withdrawn amount should be added to the `s.totalClaimed` variable

```
session.providerWithdrawnAmount += providerWithdraw;
+ s.totalClaimed += providerWithdraw;
```

Morpheus: Fixed in SessionRouter.sol#L288 and SessionRouter.sol#L532.

Renascence: Fixed. In the updated `closeSession` function, the claimed/withdrawn amount will be added to the `s.totalClaimed` variable at the end of the function when `rewardProvider` function is executed.

[H-4] Incorrect compute pool balance

Context:

- SessionRouter.sol#L413

Description: The `periodReward` is the total number of rewards earned by the compute pool from Day 0 to now (Today). `s.totalClaimed` is the total amount of rewards claimed from the compute pool by the providers up to now (Today).

So, the remaining balance of reward/MOR tokens residing on the compute pool should `periodReward - s.totalClaimed` instead of `periodReward + s.totalClaimed`.

```
File: SessionRouter.sol
401:   /// @notice returns today's compute balance in MOR
402:   function getComputeBalance(uint256 timestamp) public view returns (uint256) {
403:     Pool memory pool = s.pools[3];
404:     uint256 periodReward = LinearDistributionIntervalDecrease.getPeriodReward(
405:       pool.initialReward,
406:       pool.rewardDecrease,
407:       pool.payoutStart,
408:       pool.decreaseInterval,
409:       pool.payoutStart,
410:       uint128(startOfTheDay(timestamp))
411:     );
412:
413:     return periodReward + s.totalClaimed;
414:   }
```

The protocol's accounting mechanism relies heavily on the `getComputeBalance` function. Since the value returned from the `getComputeBalance` function is inaccurate, a wide range of issues could occur, such as the amount of stipend users are entitled to and the amount of MOR tokens required to be staked when the opening session is off.

Recommendation: Review the formula of `getComputeBalance` function to ensure that it reflects the actual use case of the protocol.

```
- return periodReward + s.totalClaimed;
+ return periodReward - s.totalClaimed;
```

Morpheus: Fixed in SessionRouter.sol#L458-L470

Renascence: The recommendation has been implemented.

[H-5] Providers receive fewer MOR tokens than expected during session closure

Context:

- SessionRouter.sol#L216

Description: The code in Lines 214-219 attempts to compute the number of MOR tokens the providers can claim if a session was closed on the same day (=not late).

```
File: SessionRouter.sol
175:   function closeSession(bytes memory receiptEncoded, bytes memory signature)
external {
..SNIP..
203:     // calculate provider withdraw
204:     uint256 providerWithdraw;
205:     bool isClosingLate = startOfTheDay(block.timestamp) >
startOfTheDay(session.endsAt);
206:     bool noDispute = isValidReceipt(session.provider, receiptEncoded, signature);
207:
208:     if (noDispute || isClosingLate) {
..SNIP..
213:     } else {
214:       // session was closed on the same day or earlier with dispute
215:       // withdraw all funds except for today's session cost
216:       uint256 durationTillToday = startOfTheDay(block.timestamp) -
217:         minUint256(session.openedAt, startOfTheDay(block.timestamp));
218:       uint256 costTillToday = durationTillToday * session.pricePerSecond;
219:       providerWithdraw = costTillToday - session.providerWithdrawnAmount;
220:     }
```

Consider the following three (3) scenarios:

Scenario 1

Assume that the session ends on Tuesday (00:00). The User closes the session 6 hours later at (06:00) on the same day (Still on Tuesday). Thus, the `isClosingLate` evaluates as `False`. The duration till today computed is 86400, which is correct.

```
block.timestamp = 1719295200 [Tue Jun 25 2024 06:00:00 GMT+0]
session.openedAt = 1719187200 [Mon Jun 24 2024 00:00:00 GMT+0]
session.endsAt = 1719273600 (1719187200 + 86400) [Tue Jun 25 2024 00:00:00 GMT+0]
isClosingLate = startOfTheDay(block.timestamp) > startOfTheDay(session.endsAt);
isClosingLate = 1719273600 > 1719273600 => False

durationTillToday = startOfTheDay(block.timestamp) - min(session.openedAt,
startOfTheDay(block.timestamp))
durationTillToday = startOfTheDay(1719295200) - min(session.openedAt,
startOfTheDay(1719295200))
durationTillToday = 1719273600 - min(1719187200, 1719273600) = 86400 (24 hours)
```

Scenario 2

Assume that the session ends on Tuesday (00:05). The User closes the session 6 hours later at (06:05) on the same day (Still on Tuesday). Thus, the `isClosingLate` evaluates as `False`. The duration till today computed is 86100, which is incorrect. The session ran for 86400 seconds, but the

provider only received 86100 seconds of fee (0.345% less).

```
block.timestamp = 1719295500 [Tue Jun 25 2024 06:05:00 GMT+0000]
session.openedAt = 1719187500 [Mon Jun 24 2024 00:05:00 GMT+0000]
session.endsAt = 1719273900 (1719187500 + 86400) [Tue Jun 25 2024 00:05:00 GMT+0000]
isClosingLate = startOfTheDay(block.timestamp) > startOfTheDay(session.endsAt);
isClosingLate = 1719273600 > 1719273600 => False

durationTillToday = startOfTheDay(block.timestamp) - min(session.openedAt,
startOfTheDay(block.timestamp))
durationTillToday = startOfTheDay(1719295500) - min(session.openedAt,
startOfTheDay(1719295500))
durationTillToday = 1719273600 - min(1719187500, 1719273600) = 86100 (23.91666667
hours)
```

Scenario 3

Assume that the session ends on Tuesday (12:00). The User closes the session 6 hours later at (18:00) on the same day (Still on Tuesday). Thus, the `isClosingLate` evaluates as `False`. The duration till today computed is 43200, which is incorrect. The session ran for 86400 seconds, but the provider only received 43200 seconds of fee (50% less).

```
block.timestamp = 1719338400 [Tue Jun 25 2024 18:00:00 GMT+0]
session.openedAt = 1719230400 [Mon Jun 24 2024 12:00:00 GMT+0]
session.endsAt = 1719316800 (1719230400 + 86400) [Tue Jun 25 2024 12:00:00 GMT+0]
isClosingLate = startOfTheDay(block.timestamp) > startOfTheDay(session.endsAt);
isClosingLate = 1719273600 > 1719273600 => False

durationTillToday = startOfTheDay(block.timestamp) - min(session.openedAt,
startOfTheDay(block.timestamp))
durationTillToday = startOfTheDay(1719338400) - min(session.openedAt,
startOfTheDay(1719338400))
durationTillToday = 1719273600 - min(1719230400, 1719273600) = 43200 (12 hours)
```

Based on the above observation, the further away from 00:00 the start time, the fewer MOR tokens the providers will receive.

Recommendation: Ensure that the number of MOR tokens the providers received is aligned with the duration of the session run. For instance, in the last scenario, the session ran for 86400 seconds. Thus, the provider should receive 86400 seconds worth of MOR tokens instead of only 43200 seconds worth of MOR tokens.

Morpheus: Acknowledged. The demonstrated bahaviour is expected.

Renascence: The issue has been acknowledged.

[H-6] Issues due to the ”rounding-down” characteristic of `startOfTheDay` function

Description: The use of the start of the day has resulted in several issues surfacing due to the ”rounding-down” characteristic of the function. The issues are regarding the `closeSession()` and `claimProviderBalance()` functions. Consider the following three (3) scenarios:

Scenario 1

Assume the ”day period” is the time points (exclusive of the right side number): `345_600 to 432_000 432_000 to 518_400`

Assume a session is `openedAt = 420_000` for a full 86400 duration, i.e. `endsAt = 506_400`

`closeSession()` is called at time point `425_000` with `noDispute=false` and `isClosingLate=false`

`durationTillToday = 345_600 - min(420_000, 345_600)` `durationTillToday = 0`

no withdraw occurs

Although the session is over, the Provider calls `claimProviderBalance()` at time point `518_400`

`claimIntervalEnd = min(518_400, 506_400)` `claimableDuration = max(506_400, 420_000) - 420_000` `claimableDuration = 86400`

Provider withdraws for full amount.

Scenarios 2 & 3

Assume the ”day periods” are:

`T24` - Day 1 `T48` - Day 2 `T72` - Day 3

Bob (user) opens a session that starts (`session.openedAt`) at T30 and ends (`session.endAt`) at T54. Note that the maximum duration is capped at 24 hours. Assume that the provider's price per second is 1 MOR (`session.pricePerSecond = 1 MOR`) for this session.

Scenario 2

At T36, Alice (provider) decided to call the `claimProviderBalance` function to claim her funds.

Let SOD = `startOfTheDay(block.timestamp)`. SOD = `startOfTheDay(T36) = T24`

```
claimIntervalEnd = min(SOD, session.endAt) = min(T24, T54) = T24

claimableDuration = max(claimIntervalEnd, session.openedAt) - session.openedAt
claimableDuration = max(T24, T30) - T30
claimableDuration = T30 - T30 = 0
```

When Alice attempts to claim her funds at T36, even though she is entitled to 6 hours' worth of funds, she receives nothing in return.

Scenario 2

Bob decided to close the session at T42 (Halfway through). `noDispute = false and isClosingLate = false`.

Let SOD = `startOfTheDay(block.timestamp)`. SOD = `startOfTheDay(T42) = T24`

```
durationTillToday = SOD - min(session.openedAt, SOD)
durationTillToday = T24 - min(T30, T24)
durationTillToday = T24 - T24 = 0

costTillToday = durationTillToday * session.pricePerSecond
costTillToday = 0 * 1 MOR
costTillToday = 0
```

Again, Alice (provider) received nothing when Bob closed the sessions halfway through the entire duration.

Recommendation: Ensure that edge cases pertaining to the rounding down to the start of days in the codebase are adequately handled.

Morpheus: The underlying issue is that the `_getProviderClaimableBalance()` function does not consider the `closedAt` field. Fixed in commit 99802bb.

Renascence: The issue has been fixed.

[H-7] There is currently no mechanism to prevent malicious Providers from earning MOR tokens at a zero cost

Context:

- SessionRouter.sol
- ModelRegistry.sol
- ProviderRegistry.sol
- Marketplace.sol

Description: A malicious actor can register as a Provider in `ProviderRegistry.providerRegister()` and register a model in `ModelRegistry.modelRegister()`. He can then call `Marketplace.postModelAgentBid()` for the created provider and model, create a signature, and call `SessionRouter.openSession()`. Although all the registrations and `openSession()` require MOR capital, there is no existing penalty or slashing mechanism that can compromise the staked funds of the malicious actor. The malicious actor proceeds to earn MOR from the `Compute` pool without providing any evidence of computational resources. The worst-case scenario for the malicious actor is that he is de-registered as a Provider by the `admin` and has his model de-registered; however, he still keeps and gets transferred his stake.

Recommendation: Either registration of Providers should be centralized, where the owner adds Providers that have a proven track record, or there should be a slashing mechanism that compromises a Provider's registration and session stake in the case of malicious activity.

Morpheus: The system benefits from the amount of costs the user staked.

Renascence: The finding has been acknowledged as a design decision.

[H-8] Unexpected revert can occur in `SessionRouter.closeSession()`.

Context:

- SessionRouter.sol#L230-L243
- SessionRouter.sol#L242

Description: If a user starts a session very close to the end of a day - `D1`, their `endsAt`, downstream `SessionRouter.openSession()` and further `SessionRouter.whenSessionEnds()`, is calculated using that day's `SessionRouter.getComputeBalance()` and `SessionRouter.totalMORSupply()`. If the user or provider wants to close the session with `isClosingLate=false` at a time point that is close to the end of the new day `D2` (and the session), the `SessionRouter.stipendToStake()` function considers `startOfTheDay(block.timestamp)`, which will be the current day, i.e., `D2`. Depending on the new amounts of `getComputeBalance()` and `totalMORSupply()`, the value of `userStakeToLock` in `SessionRouter.closeSession()` might exceed `session.stake`, causing the line `uint256 userWithdraw = session.stake - userStakeToLock;` to revert due to underflow.

Recommendation: Consider supplying the date that the session was opened to `SessionRouter.stipendToStake()` to use the pool amounts that were computed at the time of opening the session.

Morpheus: Fixed in commit 4a7067e.

Renascence: The variable `userStakeToLock` is now capped at `session.stake`, therefore `session.stake - userStakeToLock` cannot underflow.

Medium Risk

[M-1] Sessions signatures are vulnerable to cross-chain replay attacks

Context:

- SessionRouter.sol#L305

Description: Per the documentation, it was understood that the contracts are intended to be deployed to multiple blockchains.

However, it was observed that the Chain ID is not encoded/included in the signature payload for the `openSession` and `closeSession` functions. Thus, the signature obtained from a blockchain (e.g., Ethereum) can be reused across different blockchains (e.g., Polygon, Optimism).

Recommendation: The chain ID should be encoded in the signature payload and validated against the current chain ID where the action is executed. Refer to here for more information.

Morpheus: Fixed in SessionRouter.sol#L125 and SessionRouter.sol#L198.

Renascence: The recommendation has been implemented.

[M-2] Malicious user can open a session without providers approval by front-running

Context:

- SessionRouter.sol#L115

Description: The signature and `providerApproval` contains no information regarding the person to whom the approval is granted. Thus, anyone can observe the mempool, front-run the users, and use them to open a session without provider's approval.

Assume that Alice (provider) approves Bob (user) to open a session. Bob submit a `openSession` transaction to the mempool with the signature and `providerApproval` provided by Alice.

Charles saw the transaction on the mempool, and steal the signature and `providerApproval` to open a new session himself. When Bob's transaction is executed, it will revert because the signature/`providerApproval` has been marked as "spent" due to the code `s.approvalMap[providerApproval] = true;`.

```
File: SessionRouter.sol
107:   function openSession(
108:     uint256 _stake,
109:     bytes memory providerApproval,
110:     bytes memory signature
111:   ) external returns (bytes32 sessionId) {
112:     address sender = msg.sender;
113:
114:     // reverts without specific error if cannot decode abi
115:     (bytes32 bidId, uint128 timestampMs) = abi.decode(providerApproval,
(bytes32, uint128));
```

Recommendation: Consider including the intended user in the signature and check against `msg.sender` within the `SessionRouter.openSession` function.

Morpheus: Fixed in SessionRouter.sol#L112-L124.

Renascence: The recommendation has been implemented.

[M-3] Provider can DOS users by closing the session prematurely

Context:

- SessionRouter.sol#L187

Description: The session can be closed by either the user or provider. Technically, both parties have the right to terminate a session. However, in the current implementation, if a provider can close the session, this will result in unfairness for the user.

Assume that Bob (user) stakes 1000 MOR, intending to run the session for 6 hours. Shortly after Bob opens the session, Alice (provider) closes it. As a result, Bob locks 1000 MOR in the system for a day, but no work is being carried out.

```
File: SessionRouter.sol
175:   function closeSession(bytes memory receiptEncoded, bytes memory signature)
external {
..SNIP..
187:     if (session.user != msg.sender && session.provider != msg.sender) {
188:       revert NotUserOrProvider();
189:     }
```

Recommendation: Consider only allowing users to close their session.

Morpheus: Fixed by allowing only user and admin to close the session. SessionRouter.sol#L210.

Renascence: The recommendation has been implemented.

[M-4] Staked MOR tokens will be locked for longer than expected

Context:

- SessionRouter.sol#L233

Description: If the session was closed on the same day, Line 230-239 locks today's stake, so the user will not get the reward twice.

```
File: SessionRouter.sol
175:   function closeSession(bytes memory receiptEncoded, bytes memory signature)
external {
..SNIP..
203:     // calculate provider withdraw
204:     uint256 providerWithdraw;
205:     bool isClosingLate = startOfTheDay(block.timestamp) >
startOfTheDay(session.endsAt);
206:     bool noDispute = isValidReceipt(session.provider, receiptEncoded, signature);
..SNIP..
228:     // we have to lock today's stake so the user won't get the reward twice
229:     uint256 userStakeToLock = 0;
230:     if (!isClosingLate) {
231:       // session was closed on the same day
232:       // lock today's stake
233:       uint256 todaysDuration = minUint256(session.endsAt, block.timestamp) -
234:         maxUint256(startOfTheDay(block.timestamp), session.openedAt);
235:       uint256 todaysCost = todaysDuration * session.pricePerSecond;
236:       userStakeToLock = stipendToStake(todaysCost,
startOfTheDay(block.timestamp));
237:       s.userOnHold[session.user].push(
238:         OnHold({ amount: userStakeToLock, releaseAt: uint128(block.timestamp + 1
days) })
239:       );
240:     }
```

Consider the following scenario:

```
block.timestamp = 1719338400 [Tue Jun 25 2024 18:00:00 GMT+0]
session.openedAt = 1719230400 [Mon Jun 24 2024 12:00:00 GMT+0]
session.endsAt = 1719316800 (1719230400 + 86400) [Tue Jun 25 2024 12:00:00 GMT+0]
isClosingLate = startOfTheDay(block.timestamp) > startOfTheDay(session.endsAt);
isClosingLate = 1719273600 > 1719273600 => False
```

In the above example, the session ends on Tuesday (12:00). The User closes the session 6 hours later at (18:00) on the same day (Still on Tuesday). Thus, the `isClosingLate` evaluates as `False`. When `isClosingLate` is `False`, the code will attempt to lock a portion of the staked MOR for one additional day before releasing the remaining staked MOR. The number of staked MOR to be locked for one additional day is computed via the following formula:

```
todaysDuration = min(session.endsAt, block.timestamp) -
max(startOfTheDay(block.timestamp), session.openedAt);
todaysDuration = min(1719316800, 1719338400) - max(startOfTheDay(1719338400),
1719230400);
todaysDuration = min(1719316800, 1719338400) - max(1719273600, 1719230400);
todaysDuration - 1719316800 - 1719273600 = 43200

todaysCost = todaysDuration * session.pricePerSecond
```

Here, we can see that the session has already ended, and it has been 6 hours since it ended. Thus, the user's staked MOR has already been staked for more than 1 day. In fact, it has already been staked for 30 hours (24 + 6 hours) in this example.

Although the users have staked for 30 hours, the code still attempts to stake a portion of the staked MOR for another 24 hours since `isClosingLate = False`, which is incorrect and unnecessary. This will bring the duration that the users need to stake to 54 hours.

Recommendation: Ensure that the staked tokens are locked longer only if they have not been staked for 24 hours. Staked tokens should not be locked for more than 24 hours.

Morpheus: Fixed in commit 4351f8b.

Renascence: The recommendation has been implemented.

[M-5] Re-registering of a model leads to corrupt staked funds accounting

Context:

- ModelRegistry.sol#L105-L114
- ModelRegistry.sol#L67-L101

Description: When a model is de-registered through `ModelRegistry.modelDeregister()`, the `Model.stake` is not reset although the stake amount is transferred to the model owner. If the model is registered again through `ModelRegistry.modelRegister()`, the newly supplied stake is added to the existing stake; however, only the amount `addStake` is transferred to the `ModelRegistry` contract.

- Alice calls `modelRegister()`, `addStake = 100`, `Model.stake = 100`
- Alice calls `modelDeregister()`, retrieves the stake, `Model.stake = 100`
- Alice calls `modelRegister()`, `addStake = 100`, `Model.stake = 200`

Currently, in the event of a re-register, Alice cannot withdraw her funds due to `s.activeModels` not containing the model, which will revert the call to `ModelRegistry.modelDeregister()`. The out-of-sync state is also dangerous in the event re-registering is supported and the model is added again to `s.activeModels`. If the stake is not reset upon `modelDeregister()`, the funds in `ModelRegistry` can be compromised due to Alice potentially withdrawing a larger stake than deposited.

Recommendation: If re-registering is intended to be supported, then `Model.stake` should be reset upon `ModelRegistry.modelDeregister()`. Furthermore, the model should be added to `s.activeModels` but not added again to `s.models`. If re-registering is not intended to be supported, then re-registering should be prevented by reverting in `ModelRegistry.modelRegister()` if `model.isDeleted = true`.

Morpheus: Fixed in commit f37c403.

Renascence: The recommendation has been implemented.

[M-6] Re-registering of a provider leads to corrupt staked funds accounting.

Context:

- ProviderRegistry.sol#L64-L80
- ProviderRegistry.sol#L84-L93

Description: When a provider gets de-registered through `ProviderRegistry.providerDeregister()`, the `Provider.stake` is not reset although the stake amount is transferred to the provider. If the provider gets registered again through `ProviderRegistry.providerRegister()`, the newly supplied stake is added to the existing stake; however, only the amount `addStake` is transferred to the `ProviderRegistry` contract.

- Alice calls `providerRegister()`, `addStake = 100`, `Provider.stake = 100`
- Alice calls `providerDregister()`, retrieves the stake, `Provider.stake = 100`
- Alice calls `providerRegister()`, `addStake = 100`, `Provider.stake = 200`

Currently, in the event of a re-register, Alice cannot withdraw her funds due to `s.activeProviders` not containing the provider, which will revert the call to `ProviderRegistry.providerDregister()`. The out-of-sync state is also dangerous in the event re-registering is supported and the provider is added again to `s.activeProviders`. If the stake is not reset upon `providerDregister()`, the funds in `ProviderRegistry` can be compromised due to Alice potentially withdrawing a larger stake than deposited.

Recommendation: Disallow re-registration of a provider. Alternatively, if re-registering a provider should be supported, ensure that the provider is added to `s.activeProviders`, is not duplicated inside `s.providers`, and the accounting is reset. The issue with accounting is discussed in a separate issue.

Morpheus: Fixed in commit f37c403.

Renascence: The recommendation has been implemented.

[M-7] Providers and Models can participate in Sessions without being registered and staking MOR tokens

Providers and Models can participate in Sessions without being registered and staking MOR.

Context:

- SessionRouter.sol#L107-L173

Description: Providers and Models are required to be registered upon users bidding in `Marketplace.postModelBid()`. However, once bids are posted, the Providers and Models can de-register and reclaim their staked capital that was used to gain the responsibility to be a Provider or a Model. The issue is that, even though they are not registered anymore, they can still participate in Sessions since `SessionRouter.openSession()` does not validate if the Provider and Model are active.

Recommendation: During `SessionRouter.openSession()`, verify that the Provider and Model are still registered.

Morpheus: Fixed in commit 6a04f01.

Renascence: Deregistering as a provider or deregistering a model now requires that all of their bids are removed.

Low Risk

[L-1] ERC-1271 is not supported

Context:

- SessionRouter.sol#L305

Description: The open and close session functions depend on the `SessionRouter.isValidReceipt` to verify the signature/receipt is valid.

```
File: SessionRouter.sol
305:   function isValidReceipt(address signer, bytes memory receipt, bytes memory
signature) private pure returns (bool) {
306:     if (signature.length == 0) {
307:       return false;
308:     }
309:     bytes32 receiptHash =
MessageHashUtils.toEthSignedMessageHash(keccak256(receipt));
310:     return ECDSA.recover(receiptHash, signature) == signer;
311:   }
```

The provider authorizing the transaction could be from an EOA account or smart contract wallet (e.g., Safe). If the provider uses a smart contract wallet to sign the receipt, the `isValidReceipt` function won't be able to verify them as this function could only verify the signature provided by the EOA account as it uses `ECDSA.recover` function internally, and does not support ERC-1271.

Recommendation: Review if the protocol allows the provider to be a smart contract wallet. If so, consider using OZ's SignatureCheck library for signature verification, as it supports both ECDSA and ERC1271

Morpheus: Acknowledged.

Renascence: The issue has been acknowledged.

[L-2] Re-register of a model will flag the model as `isDeleted: false`, however, the model will not be active

Context:

- ModelRegistry.sol#L110
- ModelRegistry.sol#L97

Description: After `ModelRegistry.modelDeregister()`, a user can call `ModelRegistry.deregisterModel()` which sets `model.isDeleted = true` and removes the model from the `AppStorage.activeModels` set. The issue is that the user can again call `modelRegister()` for the same model ID. This means `isDeleted` will become `false`, although the model won't make it back in the `s.activeModels` set (assuming `s.modelMinStake > 0`).

Recommendation: Disallow re-registration of the same model ID. Alternatively, if re-registering the same model ID should be supported, ensure that the model is added to `s.activeModels`, is not duplicated inside `s.models`, and the accounting is reset. The issue with accounting is discussed in a separate issue.

Morpheus: Fixed in commit f37c403.

Renascence: The recommendation has been implemented.

[L-3] Re-registering a provider will flag the provider as `isDeleted: false`, even though the provider is not active

Context:

- ProviderRegistry.sol#L88
- ProviderRegistry.sol#L76

Description: After `ProviderRegistry.providerDeregister()`, a provider can call `ProviderRegistry.providerRegister()`, which sets `Provider.isDeleted = true` and removes the provider from the `AppStorage.activeProviders` set. The issue is that the provider can again call `providerRegister()`. This means `isDeleted` will become `false`, although the provider won't make it back into the `s.activeProviders` set (assuming that `s.providerMinStake > 0`).

Recommendation: Disallow re-registration of a provider. Alternatively, if re-registering a provider should be supported, ensure that the provider is added to `s.activeProviders`, is not duplicated inside `s.providers`, and the accounting is reset. The issue with accounting is discussed in a separate issue.

Morpheus: Fixed in commit f37c403.

Renascence: The recommendation has been implemented.

[L-4] Takeover of a de-registered model is possible under the assumption that `s.modelMinStake = 0`

Context:

- ModelRegistry.sol#L82-L84

Description: Alice calls `ModelRegistry.modelRegister()` with a 0 stake. Alice later calls `ModelRegistry.modelDeregister()`. Bob can now call `ModelRegistry.modelRegister()` with Alice's model ID and take over the model by recording himself as the `Model.owner`. Moreover, there will be duplicate entries of the model in `s.models`.

Recommendation: If re-registration of a model is intended to be supported, require that only a model's owner can re-register the model after being de-registered. If re-registration is not intended to be supported, prevent registering models that have `model.isDeleted = true`.

Morpheus: Fixed in commit f37c403.

Renascence: The introduction of the `createdAt` field to check model existence now prevents the model takeover described in this issue.

Informational

[I-1] Lack of timelock allowing protocol owner to pull assets from users wallet

Context:

- Marketplace.sol#L179
- Marketplace.sol#L154

Description: When a user creates a bid, the marketplace will pull the bid fee from the user's wallet, as shown in Line 154 below:

```
File: Marketplace.sol
119:   function postModelAgentBid(
..SNIP..
154:     s.token.transferFrom(msg.sender, address(this), s.bidFee);
155:     s.feeBalance += s.bidFee;
156:
157:     return bidId;
158:   }
```

If an excess allowance is granted to the protocol, protocol owners can front-run providers and execute `setBidFee` to set a high fee, which can result in a large number of tokens being pulled from the provider's wallet.

Recommendation: Consider implementing a timelock for any change to the protocol's parameters.

Morpheus: Acknowledged.

Renascence: The issue has been acknowledged.

[I-2] Lack of validation for delete history

Context:

- SessionRouter.sol#L298

Description: Safeguard should be in place to ensure that the session has been completely closed before allowing users to delete their sessions. Otherwise, their staked assets will be blocked.

```
File: SessionRouter.sol
297:   /// @notice deletes session from the history
298:   function deleteHistory(bytes32 sessionId) external {
299:     Session storage session = s.sessions[s.sessionMap[sessionId]];
300:     LibOwner._senderOrOwner(session.user);
301:     session.user = address(0);
302:   }
```

Recommendation: Consider implementing an additional check to ensure that the session has been completely closed before deleting the history.

Morpheus: Fixed in SessionRouter.sol#L338-L344.

Renascence: The recommendation has been implemented.

[I-3] Out of gas error due to unbounded loop

Context: SessionRouter.sol#L314

Description: The withdrawableUserStake function might encounter an out-of-gas errors if the number of items in the on-hold listing becomes huge. As a result, internal or external parties that rely on this function might revert unexpectedly.

```
File: SessionRouter.sol
314:   function withdrawableUserStake(address userAddr) external view returns
(uint256 avail, uint256 hold) {
315:     OnHold[] memory onHold = s.userOnHold[userAddr];
316:     for (uint i = 0; i < onHold.length; i++) {
317:       uint256 amount = onHold[i].amount;
318:       if (block.timestamp < onHold[i].releaseAt) {
319:         hold += amount;
320:       } else {
321:         avail += amount;
322:       }
323:     }
324:     return (avail, hold);
325:   }
```

Recommendation: Consider updating the `withdrawableUserStake` function to allow users to define the start and end positions for the items they want to read from the listing.

Morpheus: Fixed in commit 83fa8a9.

Renascence: The recommendation has been implemented.

[I-4] Code improvements

Context:

1. Marketplace.sol#L35

Description / Recommendation:

1. Instead of a second call to `providerBidsSet.count()`, use the cached count in `length`:

```
-    for (uint i = 0; i < providerBidsSet.count(); i++) {
+    for (uint i = 0; i < length; i++) {
```

Morpheus: Fixed in commit 4f6ef9a.

Renascence: The recommendation has been implemented.

4.1 Centralization Risks

The reviewed contracts `ModelRegistry.sol`, `ProviderRegistry.sol`, `Marketplace.sol`, and `SessionRouter.sol` are deployed as facets behind a Diamond Proxy. The owner of the Proxy can add, remove or substitute functions. Therefore, the owner has full control of the protocol and its assets. The owner must be fully trusted.

4.2 Systemic Risks

The SessionRouter contract interacts with the public and private pools of the Morpheus Distributor contract. Any issues that surface in the Distributor contract can affect the reported reward amounts to SessionRouter and therefore affect the distribution of Compute MOR rewards.