OpenZeppelin | security

Mantle Network Pre-Confirmation Transactions Audit

MANTLE

April 3, 2025

Table of Contents

Summary

Type	Layer 2
Timeline	From 2025-03-17 To 2025-03-25
Languages	Go

Total Issues	13 (12 resolved, 1 partially resolved)
Critical Severity Issues	1 (1 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	5 (5 resolved)
Notes & Additional Information	4 (3 resolved, 1 partially resolved)
Client Reported Issues	2 (2 resolved)

Scope

We audited the changes made in the mantlenetworkio/op-geth repository between commits d28ec49 and 6a7275c. In addition, the following individual commits were reviewed:

- Commit `a6e2562` addressing an issue related to incorrectly handling deposit transactions.
- Commit `525bca0` addressing an issue in `IsPreconfTxFrom` to support `AllPreconfs` mode.
- Commit `3cfc154` related to supporting contract creation in `AllPreconf` mode.
- Commit `45c338f` addressing the correct initialization of `depositTxs` within `UpdateOptimismSyncStatus`.
- Commit `d22e02a` addressing an issue in correctly handling metrics for new pre-confirmed transactions.

In scope were the following files:

```
core
└── txpool
    ├── journal.go
    └── txpool.go
eth
├── backend.go
└── filters
    ├── api.go
    └── filter_system.go
internal
└── ethapi
    └── api.go
miner
├── preconf_checker.go
└── worker.go
preconf
├── deposit_log.go
├── deposit_source.go
├── id.go
├── metrics.go
├── miner_config.go
├── sync_status.go
├── timed_tx_set.go
└── tx_pool_config.go
```

System Overview

The pre-confirmation transaction feature in the Mantle Network introduces a process whereby designated accounts can submit transactions that are validated and simulated before being included in a block. This mechanism reduces latency, allowing systems that cannot rely on a long block time (2 seconds) and require near-instant responses to determine transaction validity.

Pre-Confirmation Mechanism

The pre-confirmation transaction can be submitted through the API logic, which starts by subscribing to the pre-confirmation transaction response feed and adding the transaction to the pool of pending transactions. The transaction is then forwarded to a worker responsible for executing pre-confirmation, verifying its validity, and ensuring successful execution.

Once the execution results are obtained, they are returned and passed to the event feed, which subscribed parties can consume, allowing external users to monitor the status of pre-confirmation transactions. This mechanism operates almost instantaneously, providing an immediate response without waiting for the transaction to be included in a block.

Block Creation

The block creation process follows a logical sequence, starting with retrieving pending transactions through a custom function that separates pre-confirmation transactions from regular ones. Transactions are ordered based on priority: L1-to-L2 deposit transactions first, followed by pre-confirmation transactions, and finally, regular transactions. This ensures that critical operations, such as cross-chain deposits, are processed with the highest priority.

Address Whitelisting for Pre-confirmation Transactions

Pre-confirmation transactions are currently restricted to a predefined whitelist of sender addresses (Fiat24 Bank withdrawal accounts) and recipient addresses (Fiat24 withdrawal contracts). This restriction ensures controlled usage of the pre-confirmation feature, thereby

preventing unauthorized access. However, this behavior might change in the future, as the update introduces a flag that allows all transactions to be treated as pre-confirmation transactions.

Security Model and Trust Assumption

Since there are no specific plans to disable address whitelisting for pre-confirmation transactions, it was assumed for the scope of the audit that pre-confirmation transactions can only be created by authorized addresses.

In addition, it was assumed that both the L1 RPC and the Optimism Node operate correctly and in accordance with their respective specifications. Any failure, misconfiguration, or malicious behavior of these components was considered out of scope for the system security model.

Privileged Roles

The implemented changes add a privileged role that can submit pre-confirmation transactions. The lists of authorized addresses for transaction origins (`from`) and recipients (`to`) are provided as configuration parameters and separated by commas.

Critical Severity

C-01 Deposit Transactions Are Not Applied to the State for Simulation

The `preconfChecker` struct maintains a copy of the chain environment and a slice of synced deposit transactions that must be considered when simulating pre-confirmed transactions for the next block.

However, while deposit transactions are appended to the transactions slice in the environment, their state diff is never applied. As a result, when pre-confirmed transactions are simulated, they do not account for the deposit transactions that would be executed first. This can lead to a situation where a pre-confirmed transaction is simulated successfully and emitted as an event to the user, but during block sealing, a deposit transaction executed beforehand interferes with it, potentially changing its outcome from success to failure.

To ensure simulation accuracy, consider applying the state diff of deposit transactions before simulating pre-confirmed transactions. This will align the simulation environment with the actual state during block sealing and prevent inconsistencies.

Update: *Resolved in commit a6e2562.*

High Severity

H-01 Incorrect Handling of Deposit Transactions

Deposit transactions are not handled correctly within the pre-confirmation transactions implementation. This causes `preconfTxs` to not be properly removed, resulting in the `extractPreconfTxsFromPending` function returning incorrect data.

Consider redesigning the logic so that `extractPreconfTxsFromPending` correctly extracts pre-confirmed transactions from the pending list.

Update: *Resolved in commit a6e2562.*

Low Severity

L-01 Incorrect Metrics Gathering

The `Add` function in `timed_tx_set.go` replaces an existing transaction by removing it from the queue and re-adding it to the end. However, it increments metrics and logs each time without distinguishing replacements from new additions, suggesting that a new pre-confirmed transaction was added, even for updates. This could inflate pre-confirmation counts and mislead monitoring.

Consider incrementing metrics only for new transactions to accurately reflect additions.

Update: *Resolved in commit d22e02a.*

L-02 Validation of Pre-Confirmation Transaction Does Not Support `AllPreconfs` Mode

The `IsPreconfTxFrom` function validates whether the `from` address belongs to the list of authorized addresses. However, it does not take into account the `AllPreconfs` mode, which enables all transactions to be pre-confirmation transactions. In addition, the logic in `IsPreconfTx` checks if the `AllPreconfs` mode is enabled, but it does so only after verifying the `from` and `to` addresses.

Consider returning `true` for both the `IsPreconfTxFrom` and `IsPreconfTx` functions when the `AllPreconfs` mode is enabled.

Update: *Resolved in commit 525bca0.*

L-03 Missing Timeout Settings for L1 RPC and Optimism Node Connections

The connections to the L1 RPC and Optimism Node do not have defined timeout options. This could lead to issues if the server accepts the connection but fails to respond to the request.

Consider adding a timeout to external connections.

Update: *Resolved in commit 2e2875c.*

L-04 Missing Cleanup in `demoteUnexecutables`

The `demoteUnexecutables` function is invoked during a chain reorganization (reorg)—when a blockchain node switches to a new, longer chain, potentially invalidating transactions from the old chain—to clean up the mempool by removing or re-queuing transactions from the pending pool that are no longer valid due to insufficient gas, low account balance, or outdated nonces reflecting the updated state. While it removes unpayable (low balance or out-of-gas) transactions from the `all` lookup set, it does not remove them from the `preconfTxs` set, leaving pre-confirmed transactions that are no longer executable in the pool. This could lead to unnecessary re-evaluation of these transactions in future cycles, even though they are unlikely to succeed, whereas they could instead be directly removed for consistency and efficiency.

Consider removing these costly (low balance or out-of-gas) transactions from the `preconfTxs` set to ensure that only executable, pre-confirmed transactions remain.

Update: *Resolved in commit 547cdf8.*

L-05 Transactions May Be Skipped in `commitTimedTransactions`

The `commitTimedTransaction` function function is called during block construction to process a list of pre-confirmed transactions (`txs`). However, it breaks if `tx` is equal to `nil`, based on a comment suggesting that `nil` only appears at the end. While the current logic implies that `txs` should not contain nil entries, an unexpected `nil` in the middle of the list could cause the function to break early, skipping subsequent valid transactions. This could result in valid transactions being excluded from the block when they should be processed.

Consider removing the `nil` check or using `continue` instead of `break` to ensure that all valid transactions are processed.

Update: *Resolved in commit 2e2875c.*

Notes & Additional Information

N-01 Incorrect Documentation

Throughout the codebase, multiple instances of incorrect documentation were identified:

- The comment for the `SubscribePreconfTxs` function is copied from `SubscribePendingTxs`.
- The comment for the `SendRawTransactionWithPreconf` function is copied from `SendRawTransaction`.
- The comment in the `rotate` function about adding regular transactions to the journal is inaccurate.
- Several comments describing the default configuration are incorrect:
  - The `MantleToleranceDuration` is 6 seconds, not 5.
  - The `EthToleranceDuration` is 72 seconds, not 30.
  - The `EthToleranceBlock` is 6 blocks, not 5.

Consider reviewing the codebase to correct these inaccuracies to improve its overall readability and correctness.

Update: *Resolved in commits 1d76268, 2a2f613 and a6e2562.*

N-02 Unused Code

Throughout the codebase, multiple instances of unused code were identified:

- In `txpool.go`, the `handlePreconfTxs` function aggregates all non-timeout pre-confirmation transactions and returns them, but its return value is never used.
- In `deposit_log.go`, the `MarshalDepositLogEventV0` and `marshalDepositVersion0` functions are unused.
- None of the functions in `id.go` are used.

Consider removing these unused functions to improve the clarity and maintainability of the codebase.

Update: *Partially resolved in commit 2e2875c. The files* `deposit_log.go` *and* `id.go` *remained unchanged as they are a direct copy from the op-node component.*

N-03 Confusing Debugging Logs

In case adding a pre-confirmed transaction to the journal fails, the logged warnings are confusing. The logs will first indicate that the transaction failed, followed by information suggesting that the transaction was journaled.

Consider adding an `else` statement to indicate that the transaction was journaled if no error occurred.

Update: *Resolved in commit 2e2875c.*

N-04 Unused Aspect of `TimedTxSet`

`TimedTxSet` manages the order of incoming pre-confirmed transactions in a FIFO manner. However, two aspects of this struct may cause confusion:

- Its name suggests that transactions are timed for execution at a specific point in the future.
- The `addedTime` field, which tracks when a transaction was added to the set, is unused.

Consider renaming the set to something that better reflects its FIFO nature and removing the unused `addedTime` field.

Update: *Resolved in commit 1d76268.*

Client Reported

CR-01 Contract Creation Is Not Supported in `AllPreconf` Mode

The `handlePreconfTxs` function is responsible for handling pre-confirmed transactions. The issue is that it includes logic to ensure that the `To` address is not a zero address. This is problematic because contract creation transactions use the zero address for the `To` parameter

and, while contract creation is not intended for pre-confirmed transactions, enabling the `AllPreconf` mode might prevent contract creation.

Consider removing the check that the `To` address cannot be a zero address.

Update: *Resolved in commit [3cfc154].*

CR-02 The `depositTxs` Slice Is Not Initialized Correctly Within `UpdateOptimismSyncStatus`

The `depositTxs` slice within `UpdateOptimismSyncStatus` is not initialized properly. This lack of proper initialization could lead to incorrect handling of deposit transactions, potentially causing errors in transaction processing or inconsistencies in the system's state.

It is recommended to redesign the logic to ensure that the `depositTxs` slice is initialized correctly.

Update: *Resolved in commit [45c338f].*

Conclusion

While the pre-confirmed transaction feature enhances efficiency and reduces transaction latency, it also introduces new security challenges. The current implementation mitigates key risks through structured prioritization, event-driven state management, and stringent validation mechanisms. However, expanding beyond whitelisted addresses will require additional protections to prevent abuse and ensure equitable access to transaction prioritization.

The audited codebase was a work in progress, undergoing multiple code and design changes during the audit, including fixes for issues identified by both the development and auditing teams. It is highly recommended that the codebase undergoes an extensive testing phase and another round of auditing to ensure the system is secure and safe for deployment.

Communication with the Mantle team was seamless, and they were transparent in explaining the inner workings of the system. Furthermore, they promptly implemented the suggested solutions to address the vulnerabilities identified early in the audit.