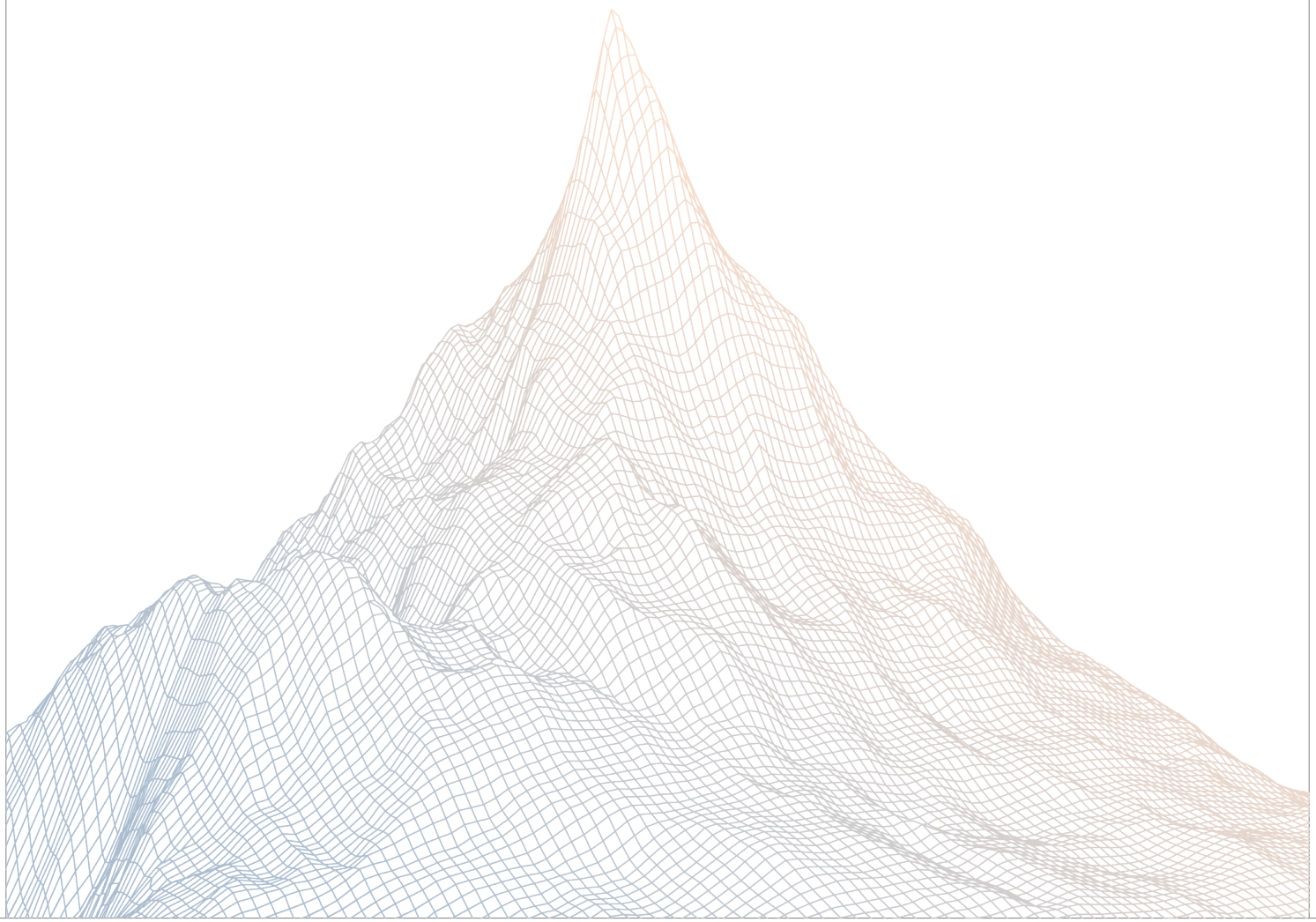
Zenith

Mantle

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: March 18th to March 23th, 2025

AUDITED BY: cccz
ladboy

Contents

1 Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

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

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Mantle

Mantle Network is dedicated to building an EVM-compatible scaling solution for Ethereum. This means that all contracts and tools running on Ethereum can operate on the Mantle Network with minimal modifications. Taking advantage of its modular architecture, Mantle Network combines an optimistic rollup with various innovative data availability solutions, providing cheaper and more accessible data availability while inheriting the security of Ethereum.

Our protocol design philosophy aims to offer users a less costly and more user-friendly experience, provide developers with a simpler and more flexible development environment, and deliver a comprehensive set of infrastructure for the next wave of mass-adopted dApps.

2.2 Scope

The engagement involved a review of the following targets:

Target	op-geth
Repository	https://github.com/mantlenetworkio/op-geth
Commit Hash	`3d54b5139f1d5ba06a40194fb4aef3f18eb00378`
Files	`Diff in PR-103`

2.3 Audit Timeline

March 18, 2025	Audit start
March 23, 2025	Audit end
March 27, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	2
Medium Risk	1
Low Risk	3
Informational	1
Total Issues	**7**

3

Findings Summary

ID	Description	Status
H-1	The node will not get depositTx when it starts	Resolved
H-2	The code cannot parse deposit transaction if other event is emitted within block range	Resolved
M-1	Preconf tx is not removed when the regular tx is removed in txPool.go	Resolved
L-1	The toBlock used when getting DepositTxs from the log is different than expected	Resolved
L-2	addPreconfTx() will not add preconf transactions with tx.To() == nil	Resolved
L-3	Avoid hardcode the timeout to 1 second when sending the deposit	Acknowledged
I-1	ctx.Err() error should be logged instead of err when encountering preconf event subscription error	Resolved

4

Findings

4.1 High Risk

A total of 2 high risk findings were identified.

[H-1] The node will not get depositTx when it starts

SEVERITY: High	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: High

Target

- preconf_checker.go#L133-L166

Description:

The protocol syncs the optimistic status to get the L1 to L2 depositTxs and inserts them at the front of the transactions.

The problem here is that `c.optimismSyncStatus` is set directly to `newOptimismSyncStatus` when the node is started, which leads to the fact that in subsequent checks, since `c.optimismSyncStatus` is equal to `newOptimismSyncStatus`, `GetDepositTxs()` will not be called to get any depositTxs, breaking the design of the protocol.

```
if c.optimismSyncStatus == nil {
    c.optimismSyncStatus = newOptimismSyncStatus
}
...
// update deposit txs if current_l1.number or head_l1.number is changed and
    optimism sync status is ok
if c.optimismSyncStatusOk {
    // update deposit txs if current_l1.number or head_l1.number is changed
    if c.optimismSyncStatus.CurrentL1.Number
    ≠ newOptimismSyncStatus.CurrentL1.Number ||
        c.optimismSyncStatus.HeadL1.Number
    ≠ newOptimismSyncStatus.HeadL1.Number {
```

Recommendations:

It is recommended to call `GetDepositTxs()` to update `c.DepositTxs` during initialization.

Mantle: Resolved with @45c338ffc51...

Zenith: Verified.

[H-2] The code cannot parse deposit transaction if other event is emitted within block range

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: High

Target

- preconf_checker

Description:

```
logs, err := c.l1ethclient.FilterLogs(context.Background(),
    ethereum.FilterQuery{
        FromBlock:  big.NewInt(int64(start)),
        ToBlock:    big.NewInt(int64(end)),
        Addresses:
    []common.Address{common.HexToAddress(c.minerConfig.L1DepositAddress)},
    })

    if err != nil {
        return nil, fmt.Errorf("failed to filter logs: %w", err)
    }

    depositTxs := make([]*types.Transaction, 0)

    for _, log := range logs {
        depositTx, err := preconf.UnmarshalDepositLogEvent(&log) // @audit
    finalize event excluded?
        if err != nil {
            return nil, fmt.Errorf("failed to unmarshal deposit log event:
    %w", err)
        }
        depositTxs = append(depositTxs, types.NewTx(depositTx))
    }
```

In the `preconf_checker`, the code extract all log from the deposit contract address and then unmarshall the deposit log.

The code assume that the smart contract only emit the deposit event in the format below.

```
// event TransactionDeposited(
//     address indexed from,
//     address indexed to,
//     uint256 indexed version,
//     bytes opaqueData
// );
```

However, the smart contract does emit other event.

Example:

TX

this tx emit the event below as well.

```
MNTBridgeInitiated (index_topic_1 address from, index_topic_2 address to,
    uint256 amount, bytes extraData)
```

Then the code will fail to parse any deposit events in the block range because the code `return` instead of `continue` when encounter parsing error.

```
for _, log := range logs {
        depositTx, err := preconf.UnmarshalDepositLogEvent(&log) // @audit
    finalize event excluded?
        if err != nil {
            return nil, fmt.Errorf("failed to unmarshal deposit log event:
    %w", err)
        }
        depositTxs = append(depositTxs, types.NewTx(depositTx))
    }
```

Parsing the `MNTBridgeInitiated` event using `UnmarshalDepositLogEvent` would get the error

```
fmt.Errorf("invalid deposit event selector: %s, expected %s", ev.Topics[0],
    DepositEventABIHash)
```

Recommendations:

```
logs, err := client.l1ethclient.FilterLogs(context.Background(),
    ethereum.FilterQuery{
        FromBlock: big.NewInt(int64(start)),
```

```
        ToBlock:   big.NewInt(int64(end)),
        Addresses: []com-
    mon.Address{common.HexToAddress(client.minerConfig.L1DepositAddress)},
.            Topics:    [][]common.Hash{{depositEventSig}},
    })
```

`depositEventSig` is the Keccak-256 hash of the event signature string for the `TransactionDeposited` event — and it's what Ethereum uses as topics[0] in logs to identify which event was emitted.

```
// event TransactionDeposited(
//     address indexed from,
//     address indexed to,
//     uint256 indexed version,
//     bytes opaqueData
// );
```

Mantle: Resolved with @547cdf8c72b...

Zenith: Verified.

4.2 Medium Risk

A total of 1 medium risk findings were identified.

[M-1] Preconf tx is not removed when the regular tx is removed in `txPool.go`

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- TxPool.go

Description:

In `txpool.go`,

the code below remove both regular tx and preconf tx.

```
func (pool *TxPool) demoteUnexecutables() {
    // Iterate over all accounts and demote any non-executable transactions
    for addr, list := range pool.pending {
        nonce := pool.currentState.GetNonce(addr)

        // Drop all transactions that are deemed too old (low nonce)
        olds := list.Forward(nonce)
        for _, tx := range olds {
            hash := tx.Hash()
            pool.all.Remove(hash)
            pool.preconfTxs.Remove(hash)
            log.Trace("Removed old pending transaction", "hash", hash)
        }
```

However, sometimes when

```
pool.all.Remove(hash)
```

is called, the code

```
pool.preconfTxs.Remove(hash)
```

is not called.

Example:

- Case 1: `func (pool *TxPool) truncatePending()` does not remove the preconf tx.
- Case 2: regular `removeTx` does not remove the preconf tx.

Recommendations:

Remove both preconf tx and regular tx in `txPool.go`

Mantle: Resolved with this commit

Zenith: Verified.

4.3 Low Risk

A total of 3 low risk findings were identified.

[L-1] The `toBlock` used when getting DepositTxs from the log is different than expected

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- preconf_checker.go#L169-L170

Description:

When fetching DepositTxs from the log, the protocol expects to use a `toBlock` two blocks away from the L1 Head to prevent reorg.

```
// c.optimismSyncStatus.CurrentL1.Number is already derived, so we need
    using c.optimismSyncStatus.CurrentL1.Number+1
// the end is not included, and we remain two block to prevent reorg, so we
    need using c.optimismSyncStatus.HeadL1.Number-1
depositTxs, err := c.GetDepositTxs(c.optimismSyncStatus.CurrentL1.Number+1,
    c.optimismSyncStatus.HeadL1.Number-1)
```

The actual `toBlock` used is `optimismSyncStatus.HeadL1.Number-1`, which is only one block away from the L1 Head.

This is because in `filterLogs()` the `toBlock`'s logs are actually fetched instead of being skipped.

```
func filterLogs(logs []*types.Log, fromBlock, toBlock *big.Int, addresses
    []common.Address, topics [][]common.Hash) []*types.Log {
  var ret []*types.Log
Logs:
  for _, log := range logs {
    if fromBlock ≠ nil && fromBlock.Int64() ≥ 0 && fromBlock.Uint64()
    > log.BlockNumber {
```

```
        continue
    }
    if toBlock ≠ nil && toBlock.Int64() ⩾ 0 && toBlock.Uint64()
    < log.BlockNumber {
        continue
    }
```

Recommendations:

```
depositTxs, err := c.GetDepositTxs
(c.optimismSyncStatus.CurrentL1.Number+1, c.optimismSyncStatus.HeadL1.
    Number-1)
depositTxs, err := c.GetDepositTxs
(c.optimismSyncStatus.CurrentL1.Number+1, c.optimismSyncStatus.HeadL1.
    Number-2)
```

Mantle: Resolved with @a6e2562e2f4...

Zenith: Verified.

[L-2] `addPreconfTx()` will not add preconf transactions with tx.To() == nil

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- txpool.go#L1233-L1240

Description:

For contract creation transactions with `tx.To() == nil`, when `AllPreconfs == true`, `IsPreconfTx()` will be true, that is, the transaction will be considered a preconf transaction.

```
func (c *TxPoolConfig) IsPreconfTx(from, to *common.Address) bool {
    // If AllPreconfs is true, all transactions are considered preconf
    if c.AllPreconfs {
        return true
    }

    if from == nil || to == nil {
        return false
    }
```

However, `addPreconfTx()` will not add them to `pool.preconfTxs`.

```
func (pool *TxPool) addPreconfTx(tx *types.Transaction) {
    txHash := tx.Hash()

    // check tx is preconf tx
    if tx.To() == nil {
        log.Debug("preconf address is nil", "tx", tx.Hash().Hex())
        return
    }
    from, _ := types.Sender(pool.signer, tx)
    if !pool.config.Preconf.IsPreconfTx(&from, tx.To()) {
        log.Debug("preconf from and to is not match", "tx", txHash)
        return
    }
```

Recommendations:

```
func (pool *TxPool) addPreconfTx(tx *types.Transaction) {
    txHash := tx.Hash()

    // check tx is preconf tx
    if tx.To() == nil {
        log.Debug("preconf address is nil", "tx", tx.Hash().Hex())
        return
    }
    from, _ := types.Sender(pool.signer, tx)
    if !pool.config.Preconf.IsPreconfTx(&from, tx.To()) {
        log.Debug("preconf from and to is not match", "tx", txHash)
        return
    }
```

Mantle: Resolved with @3cfc154ddf...

Zenith: Verified.

[L-3] Avoid hardcode the timeout to 1 second when sending the deposit

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- api.go

Description:

When sending raw pre-conf transaction, the timeout is set to 1 sec. This setting is ok for testing, but it is too short for production.

```
func (s *TransactionAPI) SendRawTransactionWithPreconf(ctx context.Context,
    input hexutil.Bytes) (*PreconfTransactionResult, error) {
    preconfTxCh := make(chan core.NewPreconfTxEvent, 100)
    defer close(preconfTxCh)
    sub := s.b.SubscribeNewPreconfTxEvent(preconfTxCh)
    defer sub.Unsubscribe()

    txHash, err := s.SendRawTransaction(ctx, input)
    if err ≠ nil {
        return nil, err
    }

    timeout := time.NewTimer(1 * time.Second)
    defer timeout.Stop()
```

Recommendations:

Recommendate the read the timeout from configuration instead of hardcode the timeout to 1 second.

Mantle: Acknowledged

4.4 Informational

A total of 1 informational findings were identified.

[I-1] `ctx.Err()` error should be logged instead of `err` when encountering preconf event subscription error

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- backend.go

Description:

```
// setup preconf subscription
event.ResubscribeErr(time.Minute, func(_ context.Context, lastErr error)
    (event.Subscription, error) {
preconfCh := make(chan core.NewPreconfTxEvent, txChanSize)
sub, err := eth.seqWebsocketService.Subscribe(ctx, "eth", preconfCh,
    "newPreconfTransaction")
if err ≠ nil {
    log.Error("Subscribe newPreconfTransaction failed", "error", err)
    return nil, err
}

// Handle received messages
go func() {
    defer sub.Unsubscribe()
    defer close(preconfCh)
    for {
        select {
        case <-ctx.Done():
            log.Error("Preconf resubscribe context error", "error", err)
            return
```

Note the code:

```
case <-ctx.Done():
    log.Error("Preconf resubscribe context error", "error", err)
    return
```

However, the error is already logged in the code above.

```
if err ≠ nil {
    log.Error("Subscribe newPreconfTransaction failed", "error", err,
    return nil, err
}
```

Recommendations:

```
case <-ctx.Done():
log.Error
("Preconf resubscribe context error", "error", err)
log.Error
("Preconf resubscribe context error", "error",
ctx.Err(), "lastErr", lastErr)
    return
```

Mantle: Resolved with @547cdf8c72bf...

Zenith: Verified.