



Security Review For Mantle



Collaborative Audit Prepared For:
Lead Security Expert(s):

Mantle
blockdev
EV_om
Koolex

Date Audited:

July 2 - August 22, 2025

Introduction

Mantle Network is an EVM-compatible scaling solution for Ethereum built as a ZK Validity Layer 2 (L2). All contracts and tools that run on Ethereum can be used on Mantle with minimal modification. Leveraging its modular architecture, Mantle separates execution, data availability, proof generation, and settlement into independent modules. Mantle now uses ZK validity proofs together with innovative data availability solutions to provide lower-cost and more accessible data availability while fully inheriting Ethereum's security guarantees.

This security review is comprised of two parts. The first part focuses on the Pectra / Skadi scope while the second is focused on Mantle's OP Succinct one.

Scope

Repository: [mantlenetworkio/mantle-v2](https://github.com/mantlenetworkio/mantle-v2)

Audited Commit: [1323d49726a8025b2a766320db293077ecfb42a3](https://github.com/mantlenetworkio/mantle-v2/commit/1323d49726a8025b2a766320db293077ecfb42a3)

Final Commit: [87af1235ec4ee75a4fe8dab2ed03213a323fe095](https://github.com/mantlenetworkio/mantle-v2/commit/87af1235ec4ee75a4fe8dab2ed03213a323fe095)

Files:

- [gas-oracle/flags/flags.go](#)
- [gas-oracle/main.go](#)
- [gas-oracle/metrics/handler.go](#)
- [gas-oracle/metrics/metrics.go](#)
- [gas-oracle/oracle/config.go](#)
- [gas-oracle/oracle/token_ratio.go](#)
- [op-batcher/batcher/batch_submitter.go](#)
- [op-batcher/batcher/channel_manager.go](#)
- [op-batcher/batcher/config.go](#)
- [op-batcher/batcher/driver_da.go](#)
- [op-batcher/batcher/driver.go](#)
- [op-batcher/cmd/main.go](#)
- [op-batcher/compressor/cli.go](#)
- [op-batcher/flags/flags.go](#)
- [op-batcher/metrics/metrics.go](#)
- [op-batcher/metrics/noop.go](#)
- [op-batcher/rpc/config.go](#)

- op-chain-ops/deployer/deployer.go
- op-chain-ops/genesis/genesis.go
- op-chain-ops/genesis/helpers.go
- op-chain-ops/genesis/layer_one.go
- op-chain-ops/genesis/layer_two.go
- op-chain-ops/state/memory_db.go
- op-heartbeat/Dockerfile
- op-heartbeat/.gitignore
- op-heartbeat/Makefile
- op-node/chaincfg/chains.go
- op-node/cmd/batch_decoder/main.go
- op-node/cmd/batch_decoder/reassemble/reassemble.go
- op-node/cmd/genesis/cmd.go
- op-node/cmd/main.go
- op-node/cmd/p2p/cmd.go
- op-node/cmd/stateviz/main.go
- op-node/flags/flags.go
- op-node/flags/p2p_flags.go
- op-node/metrics/metrics.go
- op-node/node/api.go
- op-node/node/comms.go
- op-node/node/config.go
- op-node/node/node.go
- op-node/node/runtime_config.go
- op-node/node/safedb/disabled.go
- op-node/node/safedb/safedb.go
- op-node/node/server.go
- op-node/p2p/cli/load_config.go
- op-node/p2p/cli/load_signer.go
- op-node/p2p/gossip.go
- op-node/p2p/host.go

- op-node/p2p/node.go
- op-node/p2p/rpc_api.go
- op-node/p2p/rpc_server.go
- op-node/p2p/sync.go
- op-node/rollup/da/config.go
- op-node/rollup/da/datastore.go
- op-node/rollup/derive/attributes.go
- op-node/rollup/derive/attributes_queue.go
- op-node/rollup/derive/batches.go
- op-node/rollup/derive/batch.go
- op-node/rollup/derive/batch_queue.go
- op-node/rollup/derive/calldata_source.go
- op-node/rollup/derive/channel_bank.go
- op-node/rollup/derive/channel.go
- op-node/rollup/derive/channel_in_reader.go
- op-node/rollup/derive/deposit_source.go
- op-node/rollup/derive/engine_consolidate.go
- op-node/rollup/derive/engine_queue.go
- op-node/rollup/derive/engine_update.go
- op-node/rollup/derive/frame_queue.go
- op-node/rollup/derive/l1_block_info.go
- op-node/rollup/derive/l1_retrieval.go
- op-node/rollup/derive/l1_traversal.go
- op-node/rollup/derive/l2block_util.go
- op-node/rollup/derive/payloads_queue.go
- op-node/rollup/derive/payload_util.go
- op-node/rollup/derive/pipeline.go
- op-node/rollup/derive/skadi_upgrade_transactions.go
- op-node/rollup/derive/system_config.go
- op-node/rollup/derive/test/random.go
- op-node/rollup/driver/conf_depth.go

- op-node/rollup/driver/driver.go
- op-node/rollup/driver/l1_state.go
- op-node/rollup/driver/metered_engine.go
- op-node/rollup/driver/metered_l1fetcher.go
- op-node/rollup/driver/origin_selector.go
- op-node/rollup/driver/sequencer.go
- op-node/rollup/driver/state.go
- op-node/rollup/iface.go
- op-node/rollup/output_root.go
- op-node/rollup/sync/start.go
- op-node/rollup/types.go
- op-node/service.go
- op-node/service_mantle.go
- op-node/sources/caching/cache.go
- op-node/sources/engine_client.go
- op-node/sources/eth_client.go
- op-node/sources/l1_client.go
- op-node/sources/l2_client.go
- op-node/sources/receipts.go
- op-node/sources/rollupclient.go
- op-node/sources/sync_client.go
- op-node/sources/types.go
- op-node/version/version.go
- op-proposer/cmd/main.go
- op-proposer/flags/flags.go
- op-proposer/metrics/metrics.go
- op-proposer/metrics/noop.go
- op-proposer/proposer/config.go
- op-proposer/proposer/l2_output_submitter.go
- op-service/backends/simulated.go
- op-service/cliapp/flag.go

- `op-service/cliapp/lifecycle.go`
- `op-service/crypto/signature.go`
- `op-service/ctxinterrupt/context.go`
- `op-service/ctxinterrupt/doc.go`
- `op-service/ctxinterrupt/funcs.go`
- `op-service/ctxinterrupt/signal-waiter.go`
- `op-service/ctxinterrupt/waiter.go`
- `op-service/eigenda/cli.go`
- `op-service/eigenda/da_proxy.go`
- `op-service/eth/account_proof.go`
- `op-service/eth/blob.go`
- `op-service/eth/blobs_api.go`
- `op-service/eth/block_info.go`
- `op-service/eth/heads.go`
- `op-service/eth/label.go`
- `op-service/eth/output.go`
- `op-service/eth/receipts.go`
- `op-service/eth/ssz.go`
- `op-service/eth/status.go`
- `op-service/eth/sync_status.go`
- `op-service/eth/transactions.go`
- `op-service/eth/types.go`
- `op-service/hsm/hsm_signer.go`
- `op-service/httputil/wrapped_response_writer.go`
- `op-service/log/cli.go`
- `op-service/log/defaults.go`
- `op-service/log/dynamic.go`
- `op-service/log/http.go`
- `op-service/log/writer.go`
- `op-service/metrics/cli.go`
- `op-service/metrics/doc/cmd.go`

- op-service/metrics/ref_metrics.go
- op-service/pprof/cli.go
- op-service/predeploys/eip2935.go
- op-service/predeploys/eip4788.go
- op-service/rpc/cli.go
- op-service/solabi/util.go
- op-service/testlog/capturing.go
- op-service/tls/cli.go
- op-service/txmgr/cli.go
- op-service/txmgr/txmgr.go
- op-service/util.go
- op-service/version.go
- op-signer/client/config.go

Repository: mantlenetworkio/op-geth

Audited Commit: 1710eea21878683dce27fa7b22d2b3bb107c5d56

Final Commit: 344fa46a6a4d9acc3240fc5dc43adc9210ea8132

Files:

- accounts/abi/bind/backends/simulated.go
- beacon/engine/gen_blockparams.go
- beacon/engine/gen_ed.go
- beacon/engine/gen_epe.go
- beacon/engine/types.go
- .circleci/check-releases.sh
- .circleci/ci-docker-tag-op-geth-release.sh
- .circleci/config.yml
- cmd/era/main.go
- cmd/evm/internal/t8ntool/execution.go
- cmd/geth/config.go
- cmd/geth/main.go
- cmd/geth/misc/cmd.go

- `cmd/utils/flags.go`
- `cmd/utils/flags_test.go`
- `common/types.go`
- `consensus/beacon/consensus.go`
- `consensus/beacon/oplegacy.go`
- `consensus/clique/clique.go`
- `consensus/ethash/consensus.go`
- `consensus/misc/eip1559/eip1559.go`
- `consensus/misc/eip4844/eip4844.go`
- `core/blockchain.go`
- `core/block_validator.go`
- `core/chain_makers.go`
- `core/error.go`
- `core/events.go`
- `core/evm.go`
- `core/genesis.go`
- `core/genesis_test.go`
- `core/rawdb/accessors_chain.go`
- `core/rawdb/accessors_chain_test.go`
- `core/rawdb/accessors_indexes_test.go`
- `core/rawdb/accessors_metadata.go`
- `core/rawdb/chain_iterator_test.go`
- `core/rlp_test.go`
- `core/state_prefetcher.go`
- `core/state_processor.go`
- `core/state_processor_test.go`
- `core/state/statedb.go`
- `core/state_transition.go`
- `core/state_transition_test.go`
- `core/tracing/gen_balance_change_reason_stringer.go`
- `core/tracing/hooks.go`

- core/txpool/blobpool/blobpool.go
- core/txpool/blobpool/blobpool_preconf.go
- core/txpool/errors.go
- core/txpool/legacypool/legacypool.go
- core/txpool/legacypool/legacypool_preconf.go
- core/txpool/legacypool/legacypool_test.go
- core/txpool/legacypool/list.go
- core/txpool/locals/journal.go
- core/txpool/locals/pool_journaler.go
- core/txpool/locals/preconf_tx_tracker.go
- core/txpool/subpool.go
- core/txpool/txpool.go
- core/txpool/txpool_preconf.go
- core/txpool/validation.go
- core/types/block_config.go
- core/types/block.go
- core/types/block_test.go
- core/types/deposit_tx.go
- core/types/gen_receipt_json.go
- core/types/meta_transaction.go
- core/types/meta_transaction_test.go
- core/types/receipt.go
- core/types/receipt_test.go
- core/types/rollup_cost.go
- core/types/rollup_cost_test.go
- core/types/transaction.go
- core/types/transaction_marshallig.go
- core/types/transaction_marshallig_test.go
- core/types/transaction_signing.go
- core/types/tx_access_list.go
- core/types/tx_blob.go

- core/types/tx_dynamic_fee.go
- core/types/tx_legacy.go
- core/types/tx_setcode.go
- core/verkle_witness_test.go
- core/vm/contracts.go
- core/vm/contracts_test.go
- core/vm/evm.go
- core/vm/interpreter.go
- core/vm/jump_table.go
- core/vm/runtime/runtime.go
- core/vm/testdata/precompiles/p256Verify.json
- crypto/secp256r1/publickey.go
- crypto/secp256r1/verifier.go
- crypto/secp256r1/verifier_test.go
- Dockerfile
- entrypoint.sh
- eth/api_backend.go
- eth/api_debug.go
- eth/backend.go
- eth/catalyst/api.go
- eth/catalyst/api_test.go
- ethclient/ethclient.go
- ethclient/ethclient_test.go
- ethclient/example_test.go
- ethclient/simulated/backend.go
- eth/downloader/downloader.go
- eth/downloader/queue.go
- eth/ethconfig/config.go
- eth/ethconfig/gen_config.go
- eth/filters/api.go
- eth/filters/filter_system.go

- eth/filters/filter_system_test.go
- eth/gasestimator/gasestimator.go
- eth/gasprice/gasprice.go
- eth/gasprice/optimism_gasprice.go
- eth/gasprice/optimism_gasprice_test.go
- eth/handler_eth.go
- eth/handler.go
- eth/state_accessor.go
- eth/tracers/api.go
- eth/tracers/api_test.go
- eth/tracers/internal/tracetest/calltrace_test.go
- eth/tracers/internal/tracetest/flat_calltrace_test.go
- eth/tracers/internal/tracetest/prestate_test.go
- eth/tracers/tracers_test.go
- fork.yaml
- .github/workflows/check-docker-build.yml
- .github/workflows/check-make-build.yml
- .github/workflows/pages.yaml
- .gitignore
- go.mod
- go.sum
- graphql/graphql.go
- internal/ethapi/api.go
- internal/ethapi/api_test.go
- internal/ethapi/backend.go
- internal/ethapi/simulate.go
- internal/ethapi/transaction_args.go
- internal/ethapi/transaction_args_test.go
- internal/flags/categories.go
- internal/version/version.go
- miner/miner.go

- miner/miner_preconf.go
- miner/miner_test.go
- miner/payload_building.go
- miner/payload_building_test.go
- miner/preconf_checker.go
- miner/preconf_checker_test.go
- miner/worker.go
- miner/worker_test.go
- params/config.go
- params/config_test.go
- params/mantle.go
- params/mantle_test.go
- params/protocol_params.go
- preconf/deposit_log.go
- preconf/deposit_source.go
- preconf/fifo_tx_set.go
- preconf/fifo_tx_set_test.go
- preconf/id.go
- preconf/metrics.go
- preconf/miner_config.go
- preconf/miner_config_test.go
- preconf/sync_status.go
- preconf/tx_pool_config.go
- preconf/tx_pool_config_test.go
- rpc/errors.go
- tests/preconf/config/config.go
- tests/preconf/contracts/TestERC20.sol
- tests/preconf/contracts/TestPay.sol
- tests/preconf/front_running/erc20.go
- tests/preconf/front_running/transfer.go
- tests/preconf/main.go

- tests/preconf/reorg/detect.go
- tests/preconf/sort/sort.go
- tests/preconf/stress/stress.go
- tests/state_test.go
- tests/state_test_util.go

Repository: mantle-xyz/kona commit diff (9dlace6...857afbf)

Final Commit: 2938087b79eb4113d94df445c65f08237ad45fdf

Files:

- bin/client/src/fpvm_evm/factory.rs
- bin/client/src/fpvm_evm/precompiles/provider.rs
- bin/node/src/commands/info.rs
- crates/node/engine/src/attributes.rs
- crates/node/engine/src/client.rs
- crates/node/engine/src/state/builder.rs
- crates/node/engine/src/state/core.rs
- crates/node/rpc/src/rollup.rs
- crates/node/service/src/service/core.rs
- crates/node/sources/src/runtime/loader.rs
- crates/proof/executor/src/builder/assemble.rs
- crates/proof/executor/src/builder/core.rs
- crates/proof/executor/src/builder/env.rs
- crates/proof/executor/src/test_utils.rs
- crates/proof/executor/src/util.rs
- crates/proof-interop/src/consolidation.rs
- crates/proof/proof/src/boot.rs
- crates/protocol/derive/src/attributes/stateful.rs
- crates/protocol/derive/src/sources/ethereum.rs
- crates/protocol/derive/src/stages/batch/batch_provider.rs
- crates/protocol/derive/src/stages/batch/batch_queue.rs
- crates/protocol/derive/src/stages/channel/channel_assembler.rs

- crates/protocol/derive/src/stages/channel/channel_provider.rs
- crates/protocol/derive/src/stages/channel/channel_reader.rs
- crates/protocol/derive/src/stages/frame_queue.rs
- crates/protocol/derive/src/stages/ll_traversal.rs
- crates/protocol/genesis/src/chain/config.rs
- crates/protocol/genesis/src/genesis.rs
- crates/protocol/genesis/src/lib.rs
- crates/protocol/genesis/src/rollup.rs
- crates/protocol/genesis/src/system/config.rs
- crates/protocol/hardforks/src/ecotone.rs
- crates/protocol/hardforks/src/fjord.rs
- crates/protocol/hardforks/src/interop.rs
- crates/protocol/hardforks/src/isthmus.rs
- crates/protocol/hardforks/src/test_utils.rs
- crates/protocol/interop/src/graph.rs
- crates/protocol/protocol/src/batch/core.rs
- crates/protocol/protocol/src/batch/reader.rs
- crates/protocol/protocol/src/batch/single.rs
- crates/protocol/protocol/src/deposits.rs
- crates/protocol/protocol/src/info/variant.rs
- crates/protocol/protocol/src/sync.rs
- crates/protocol/protocol/src/utils.rs
- crates/protocol/registry/src/superchain.rs
- examples/execution-fixture/src/main.rs

Repository: mantle-xyz/hydro

Final Commit: 26b5845d031c7e7e7e2e53343b1f96bf69983412

Files: entire repository.

Repository: mantle-xyz/op-succinct commit diff (530df86...5b156fa)

Final Commit: cd712b28f2315f020b6c8b00089fe0f859cde902

Files:

- programs/range/celestia/src/main.rs
- programs/range/eigenda/src/main.rs
- scripts/utils/bin/fetch_and_save_proof.rs
- utils/build/src/lib.rs
- utils/celestia/client/src/executor.rs
- utils/celestia/client/src/lib.rs
- utils/celestia/host/src/blobstream_utils.rs
- utils/celestia/host/src/host.rs
- utils/celestia/host/src/lib.rs
- utils/celestia/host/src/witness_generator.rs
- utils/client/src/client.rs
- utils/client/src/precompiles/factory.rs
- utils/client/src/precompiles/mod.rs
- utils/eigenda/client/src/executor.rs
- utils/eigenda/client/src/lib.rs
- utils/eigenda/host/src/host.rs
- utils/eigenda/host/src/lib.rs
- utils/eigenda/host/src/witness_generator.rs
- utils/elfs/src/lib.rs
- utils/host/src/fetcher.rs
- utils/host/src/host.rs
- utils/host/src/lib.rs
- utils/host/src/rollup_config.rs
- utils/proof/src/lib.rs
- validity/src/proof_requester.rs
- validity/src/proposer.rs

Repository: mantle-xyz/revm

Audited Commit: 59cbd48

Final Commit: 7412a0d6707ec96aa9ae9cb3e6cb8e975029f1f0

Files:

- crates/handler/src/handler.rs
- crates/interpreter/src/gas.rs
- crates/op-revm/src/constants.rs
- crates/op-revm/src/evm.rs
- crates/op-revm/src/fast_lz.rs
- crates/op-revm/src/handler.rs
- crates/op-revm/src/l1block.rs
- crates/op-revm/src/lib.rs
- crates/op-revm/src/transaction.rs
- crates/op-revm/src/transaction/abstraction.rs
- crates/op-revm/src/transaction/bvm_eth.rs
- crates/op-revm/src/transaction/deposit.rs
- crates/op-revm/src/transaction/error.rs
- examples/block_traces/src/main.rs

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
5	12	37

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Permanent L2 block loss due to missing block recovery in DA submission failure reset [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/297>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The `loopEigenDa` function contains a state management bug where exceeding `DaLoopRetryNum` (10) retries causes permanent L2 block loss. When DA submission fails, the `reset()` function destroys processed L2 blocks without recovery, breaking batcher continuity and causing data loss.

Vulnerability Detail

When `loopEigenDa` fails to submit all transactions within 10 retry attempts, it triggers a `reset()` that permanently destroys L2 blocks without recovery. Unlike channel timeout scenarios, DA submission failures lack block recovery mechanisms.

Detailed Flow:

Data Processing:

```
L2 blocks: s.blocks → pendingChannel.blocks → frames → daPendingTxData
```

Failure Scenario:

State: `s.blocks = []`, `pendingChannel.blocks = {block1, block2, block3}` ↓ Process frames from blocks ↓ Try 10 times: • `frame1` → Block 1001 ❌ • `frame2` → Block 1002 ❌ • `frame3` → FAIL ❌ ↓ Still have failures → Return ERROR ↓ `publishStateToMantleDA()` calls `reset()` ↓ `clearPendingChannel()` destroys `pendingChannel.blocks` ↓ **BLOCKS PERMANENTLY LOST**

Next mantleDALoop Iteration:

State: `s.blocks = [new blocks...]`, `pendingChannel = nil` ↓ `appendNextRollupData()` processes NEW blocks ↓ **LOST BLOCKS {block1, block2, block3} NEVER RECOVERED** ↓ Processing continues but creates L2 chain gap

The bug occurs because:

1. `processBlocks()` moves blocks from `s.blocks` to `pendingChannel.blocks`
2. `reset()` calls `clearPendingChannel()` without block recovery
3. Block recovery only exists for channel timeouts, not DA failures

Impact

- **L2 chain gaps:** Specific L2 blocks permanently lost from processing queue creating holes in sequence
- **Derivation pipeline corruption:** L2 nodes cannot reconstruct complete chain from incomplete L1 data

Code Snippet

```
func (l *BatchSubmitter) reset() {
    l.state.clearPendingChannel() // ← Destroys blocks without recovery
    l.state.clearMantleDAStatus()
    l.lastStoredBlock = eth.BlockID{}
}

func (s *channelManager) clearPendingChannel() {
    s.pendingChannel = nil // ← pendingChannel.blocks lost forever
    s.pendingTransactions = make(map[txID]txData)
    s.confirmedTransactions = make(map[txID]eth.BlockID)
}

// Compare: Channel timeout DOES recover blocks
func (s *channelManager) TxConfirmed(id txID, inclusionBlock eth.BlockID) {
    if s.pendingChannelIsTimedOut() {
        s.blocks = append(s.pendingChannel.Blocks(), s.blocks...) // ← Recovery
        s.clearPendingChannel()
    }
}
```

[batcher/driver_da.go#L439](#)

[batcher/channel_manager.go#L127-L128](#)

Tool Used

Manual Review

Recommendation

Add block recovery to reset function to prevent data loss.

```
func (l *BatchSubmitter) reset() {
    // Add missing block recovery before clearing channel
    if l.state.pendingChannel != nil {
        l.state.blocks = append(l.state.pendingChannel.Blocks(), l.state.blocks...)
    }
}
```

```
l.state.clearPendingChannel()
l.state.clearMantleDAStatus()
l.lastStoredBlock = eth.BlockID{}
}
```

Note: This fix prevents data loss but is not ideal as it causes reprocessing of all blocks (including those already successfully submitted to L1), leading to duplicate frame submissions and wasted costs. A better solution should track successful submissions independently to avoid both data loss and duplication

Discussion

adam-xu-mantle

@0xbok @koolexcrypto The channel timeout can also recover DA submission failures. In the scenario below, the op-node will discard frame1 and frame2 because it cannot retrieve frame3 after channel timeout. The op-batcher will resubmit new frames after a reset. Therefore, the only impact of a frame3 submission failure is that the derivation of the safe state will have to wait an additional channel timeout period.

State: s.blocks = [], pendingChannel.blocks = {block1, block2, block3} ↓ Process frames from blocks ↓ Try 10 times: • frame1 → Block 1001 ☒ • frame2 → Block 1002 ☒ • frame3 → FAIL ☒ ↓ Still have failures → Return ERROR

koolexcrypto

@adam-xu-mantle

Recovery only exists for channel timeouts, not DA failures. In case, frame3 → FAIL not because of a timeout (other reasons instead), then we would have this scenario.

Issue H-2: OpenZeppelin M-01 audit fix incorrectly implemented breaking all EigenDA retries [RESOLVED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/306>

Summary

The EigenDA client's error handling in `DisperseBlob` method is incompatible with the batcher's retry logic, causing immediate fallback to Ethereum blobs on any dispersal error. The batcher only retries errors of type `ErrNotFound`, but `DisperseBlob` never returns this error type, effectively disabling all retry mechanisms for EigenDA dispersal failures.

Vulnerability Detail

Call Flow Trace:

```
BatchSubmitter.loopEigenDa()
  → BatchSubmitter.disperseEigenDaData()
    → EigenDAClient.DisperseBlob()
      → Returns: ErrInvalidInput | ErrNetwork (never ErrNotFound)
  → Batcher checks: !errors.Is(err, eigenda.ErrNotFound)
    → Always evaluates to true for DisperseBlob errors
  → Batcher immediately exits retry loop and falls back to Ethereum blobs
```

The batcher implements retry logic that only continues retrying when `ErrNotFound` is returned:

```
if err != nil && !errors.Is(err, eigenda.ErrNotFound) {
    l.log.Warn("unrecoverable error in disperseEigenDaData", "retry time", retry,
        ↪ "err", err)
    break
}
```

[batcher/driver_da.go#L171](#)

However, the `DisperseBlob` method never returns `ErrNotFound`. Instead, it returns:

- `ErrInvalidInput` for empty blob data
- `ErrNetwork` wrapped errors for all network failures
- `ErrNetwork` wrapped errors for all HTTP status code failures

The `ErrNotFound` error type is only returned by:

- `RetrieveBlobWithCommitment` method for HTTP 404 responses
- `GetBlobExtraInfo` method for HTTP 404 responses

Neither of these methods are called during the dispersal process that the batcher's retry logic handles.

This creates a logical mismatch where the batcher's retry condition `!errors.Is(err, eigenda.ErrNotFound)` will always evaluate to true for any `DisperseBlob` error, causing immediate retry loop termination.

Impact

Any dispersal failure immediately triggers fallback to Ethereum blob storage, including transient network issues, temporary server overload, or rate limiting that should be retried.

This contradicts the previous OpenZeppelin audit finding **"M-01 Inefficient Error Handling and Timeout in loopEigenDa Loop"** which was marked as **"Resolved in pull request #192"** with the claim that:

"The Mantle team implemented a more fine-grained distinction between transient and permanent errors. Transient errors will be retried, while permanent errors such as EigenDA invocation errors will exit immediately."

However, the actual implementation **fails to distinguish between error types** and treats all dispersal errors as permanent, making the audit fix ineffective and the retry mechanism non-functional.

Code Snippet

Batcher retry logic:

```
for retry := 0; retry < EigenRPCRetryNum; retry++ {
    wrappedData, err = l.disperseEigenDaData(daData)
    if err == nil && len(wrappedData) > 0 {
        eigendaSuccess = true
        break
    }

    if err != nil && !errors.Is(err, eigenda.ErrNotFound) {
        l.log.Warn("unrecoverable error in disperseEigenDaData", "retry time",
            ↪ retry, "err", err)
        break // Always executes for DisperseBlob errors
    }
}
```

[batcher/driver_da.go#L171](#)

DisperseBlob error returns:

```
func (c *EigenDAClient) DisperseBlob(ctx context.Context, img []byte)
    ↪ (*disperser.BlobInfo, error) {
```

```

    if len(img) == 0 {
        return nil, ErrInvalidInput // Not ErrNotFound
    }

    resp, err := c.disperseClient.Do(req)
    if err != nil {
        return nil, fmt.Errorf("failed to post store data: %w error: %w", err,
            ↳ ErrNetwork) // Not ErrNotFound
    }

    if resp.StatusCode != http.StatusOK {
        return nil, fmt.Errorf("failed to store data status code: %v error: %w",
            ↳ resp.StatusCode, ErrNetwork) // Not ErrNotFound
    }
}

```

[op-service/eigenda/da_proxy.go#L135](#)

ErrNotFound only used in retrieval methods:

```

func (c *EigenDAClient) RetrieveBlobWithCommitment(ctx context.Context, commitment
↳ []byte) ([]byte, error) {
    if resp.StatusCode == http.StatusNotFound {
        return nil, ErrNotFound // Only used in retrieval, not dispersal
    }
}

func (c *EigenDAClient) GetBlobExtraInfo(ctx context.Context, commitment []byte)
↳ (map[string]interface{}, error) {
    if resp.StatusCode == http.StatusNotFound {
        return nil, ErrNotFound // Only used in retrieval, not dispersal
    }
}

```

[eigenda/da_proxy.go#L114](#) [eigenda/da_proxy.go#L222](#)

Tool Used

Manual Review

Recommendation

Update the batcher retry condition to retry ErrNetwork errors:

```

if err != nil && !errors.Is(err, eigenda.ErrNetwork) {
    l.log.Warn("unrecoverable error in disperseEigenDaData", "retry time", retry,
        ↳ "err", err)
}

```

```
    break  
}
```

This allows the batcher to retry transient network failures and server errors that are currently wrapped with `ErrNetwork`, while immediately stopping on permanent errors like `ErrInvalidInput`.

Discussion

adam-xu-mantle

Addressed in this pr [#222](#)

Issue H-3: BVM_ETH value forwarding persists on REVERT [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/458>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Deposit-time BVM_ETH forwarding is executed before EVM execution and persists on runtime REVERT, which can irreversibly strand tokens when the deposit targets CREATE.

Vulnerability Detail

In the deposit pre-execution path, `validate_against_state_and_deduct_caller()` applies BVM_ETH operations for deposits ahead of the interpreter's checkpoint/revert lifecycle. It calls `BvmEth::mint()` and then `BvmEth::transfer()` (for `eth_tx_value`) during validation, before any frame is created.

Runtime REVERT only rolls back changes performed within the interpreter-managed checkpoint (see `EthFrame::process_next_action()` and `return_create()/return_call()`), not changes applied in pre-exec. The deposit error path (`catch_error()`) clears the journal on transaction-level failure, but runtime REVERT does not go through that path. Consequently, `eth_tx_value` forwarding persists on REVERT, updating balances and emitting Transfer logs despite the failed execution.

Persisting native MNT credit and the base BVM_ETH mint (`eth_value`) on failure aligns with deposit semantics. The problem is pre-exec forwarding of `eth_tx_value`, which is semantically a runtime transfer and should be atomic with execution.

Impact

A deposit whose execution reverts can still forward BVM_ETH (`eth_tx_value`) and emit ERC20 Transfer logs, creating observable balance and event changes despite a failed transaction. If a deposit targets CREATE and the execution reverts, the pre-exec BVM_ETH forwarding can transfer tokens to the derived CREATE address even though no contract is deployed there, rendering the tokens irrecoverable because that address has no code and no private key.

Code Snippet

```
/* crates/op-revm/src/handler.rs */
fn validate_against_state_and_deduct_caller(&self, evm: &mut Self::Evm) ->
    Result<(), Self::Error> {
```

```

let ctx = evm.ctx();
let is_deposit = ctx.tx().tx_type() == DEPOSIT_TRANSACTION_TYPE;

if is_deposit {
    if let Some(eth_value) = ctx.tx().eth_value() {
        BvmEth::mint(ctx, U256::from(eth_value)).map_err(ERROR::from)?;
    }
    if let Some(eth_tx_value) = ctx.tx().eth_tx_value() {
        BvmEth::transfer(ctx, U256::from(eth_tx_value)).map_err(ERROR::from)?;
    }
}

/* ... remainder omitted ... */
}

```

Tool Used

Manual Review

Recommendation

- Keep deposit mint semantics: native MNT credit and base BVM_ETH mint (eth_value) should continue to persist on failure if that is the intended deposit rule.
- Only gate and stage the forwarding path: wrap eth_tx_value in a journal checkpoint created before execution, make it visible during execution, and commit on success or revert on REVERT. For CREATE, derive the target without mutating nonce and forward only under this checkpoint.

Discussion

RealicZ

Addressed in this pr <https://github.com/mantle-xyz/revm/pull/10>

OxEVom

@RealicZ while <https://github.com/mantle-xyz/revm/pull/10> allows recovering the forwarded tokens after a runtime REVERT by immediately executing a successful CREATE transaction, forwarding still happens pre-exec and persists on REVERT. If the caller's nonce changes before a successful CREATE, the tokens remain stranded. The PR does not stage the transfer under a reversible checkpoint as recommended and the risk of loss of funds remains.

RealicZ

This part of the logic is consistent with op-geth. We'll consider adjusting the REVERT order in a future version.

https://github.com/mantlenetworkio/op-geth/blob/v1.3.3/core/state_transition.go#L597-L606

Issue H-4: BVM_ETH forwarding for CREATE deposits targets a different address than the deployed contract [RESOLVED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/459>

Summary

For deposits targeting CREATE, pre-exec BVM_ETH forwarding (`eth_tx_value`) derives the recipient from a pre-bump nonce while the actual CREATE path bumps the nonce again, causing tokens to be sent to `create(n)` while the contract (if deployed) lives at `create(n+1)`.

Vulnerability Detail

In the deposit path, `validate_against_state_and_deduct_caller()` forwards BVM_ETH via `BvmEth::transfer()` when `eth_tx_value` is present, ahead of execution. The forwarding implementation in `BvmEth::transfer_inner()` handles `TxKind::Create` by:

- incrementing the sender nonce using `journal().inc_account_nonce(from)`,
- computing `old_nonce = nonce - 1`,
- deriving the forwarding target as `from.create(old_nonce)`.

Later, the actual CREATE path in the interpreter setup (`EthFrame::make_create_frame()` within `frame.rs`) increments the nonce again before deriving the created address:

- bumps nonce via `journal().inc_account_nonce(inputs.caller)`,
- computes `old_nonce = nonce - 1`,
- derives the created address as `inputs.caller.create(old_nonce)`.

Because a first nonce bump has already been performed by `BvmEth::transfer_inner()` prior to execution, the interpreter's bump moves the effective `old_nonce` forward by one. The BVM_ETH is forwarded to `create(n)` while the contract is deployed at `create(n+1)` on success. On revert, no contract is deployed and the forwarded tokens are sent to an undeployed address.

Impact

For CREATE deposits, BVM_ETH value can be irreversibly misrouted. On success, tokens end up at an address different from the deployed contract, breaking expected funding flows.

Code Snippet

```
/* crates/op-revm/src/transaction/bvm_eth.rs */
fn transfer_inner<CTX>(context: &mut CTX, eth_value: U256) -> Result<(),
↳ OpTransactionError> {
    let from = context.tx().caller();
    let to = match context.tx().kind() {
        TxKind::Call(address) => address,
        TxKind::Create => {
            // Increase nonce of caller and check if it overflows
            let Some(nonce) =
                ↳ context.journal().inc_account_nonce(from).map_err(db_error)? else {
                    return Err(OpTransactionError::BvmEth(BvmEthError::NonceOverflow));
                };
            let old_nonce = nonce - 1;
            from.create(old_nonce)
        }
    };

    /* ... forwarding occurs to `to` derived above ... */
}
```

```
/* crates/handler/src/frame.rs */
pub fn make_create_frame(/* ... */) -> Result<ItemOrResult<Self, FrameResult>,
↳ ERROR> {
    /* ... */
    // Increase nonce of caller and check if it overflows
    let old_nonce;
    if let Some(nonce) = context.journal().inc_account_nonce(inputs.caller)? {
        old_nonce = nonce - 1;
    } else {
        return return_error(InstructionResult::Return);
    }

    // Create address
    let created_address = match inputs.scheme {
        CreateScheme::Create => inputs.caller.create(old_nonce),
        /* ... */
    };
    /* ... */
}
```

Tool Used

Manual Review

Recommendation

Do not call `journal().inc_account_nonce()` in `BvmEth::transfer_inner()` for `TxKind::Create`. Instead, derive the expected create address without mutating state and do not forward pre-exec.

Discussion

RealicZ

Acknowledged, it indeed should not call `journal().inc_account_nonce()` in `BvmEth::transfer_inner()`. The nonce should be used directly, and this will be fixed in the PR.

```
let to = match context.tx().kind() {
  TxKind::Call(address) => address,
  TxKind::Create => {
-      // Increase nonce of caller and check if it overflows
-      let Some(nonce) = context
+      let nonce = context
          .journal()
-          .inc_account_nonce(from)
+          .load_account(from)
          .map_err(db_error)?
-      else {
-          return
-      }
-      Err(OpTransactionError::BvmEth(BvmEthError::NonceOverflow));
-      };
-      let old_nonce = nonce - 1;
-      from.create(old_nonce)
+      .data
+      .info
+      .nonce;
+      from.create(nonce)
  }
};
```

Issue H-5: Failed deposits drop BVM_ETH mint in revm [RESOLVED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/475>

Summary

On transaction-level failed deposits, revm discards the BVM_ETH mint while op-geth persists it, yielding divergent balances/total supply and a different state root.

Vulnerability Detail

In revm, deposit-side BVM_ETH side effects are applied before execution. Specifically, `validate_against_state_and_deduct_caller()` performs `BvmEth::mint` (for `eth_value`) and then `BvmEth::transfer` (for `eth_tx_value`) during pre-execution validation. If the deposit subsequently fails at the transaction level (i.e., the engine returns an outer error for the deposit), the revm error handler `catch_error()` reconstructs a minimal overlay containing only the caller's native balance (crediting the deposit mint) and `nonce`, then clears the journal. This drops all BVM_ETH storage/log updates produced pre-execution, including the mint.

In op-geth, BVM_ETH mint is also applied before execution, but crucially it occurs before a state snapshot, and only the interpreter result distinguishes VM reverts from outer transaction failure. `execute()` applies the mint and takes a snapshot before calling `innerExecute()`:

- BVM_ETH mint: [core/state_transition.go#L589-L606](#)
- Snapshot after mint: [core/state_transition.go#L604-L612](#)

On a transaction-level failed deposit (outer error), op-geth reverts to that snapshot and increments the caller nonce, which preserves the pre-snapshot mint:

- Failed-deposit revert path: [core/state_transition.go#L620-L631](#)

Net result: for the same failed deposit containing `eth_value`, op-geth persists the BVM_ETH mint while revm drops it.

Impact

A failed deposit with `eth_value` causes BVM_ETH balance and total supply to increase in op-geth but not in revm, producing a consensus-relevant state root divergence and mismatched logs for the Mint event.

Code Snippet

```
// revm: deposit pre-exec mint/transfer
if is_deposit {
    if let Some(eth_value) = ctx.tx().eth_value() {
        BvmEth::mint(ctx, U256::from(eth_value))?;
    }
    if let Some(eth_tx_value) = ctx.tx().eth_tx_value() {
        BvmEth::transfer(ctx, U256::from(eth_tx_value))?;
    }
}

// revm: transaction-level failed deposit handler drops BVM_ETH state
let account = {
    let mut acc = Account::from(evm.ctx().db().basic(caller)?.unwrap_or_default());
    acc.info.nonce = acc.info.nonce.saturating_add(1);
    acc.info.balance =
        ↪ acc.info.balance.saturating_add(U256::from(mint.unwrap_or_default()));
    acc.mark_touch();
    acc
};
let state = HashMap::from_iter([(caller, account)]);
evm.ctx().journal().clear(); // BVM_ETH storage/logs are discarded

// op-geth: mint occurs before snapshot -> persists on outer error
if ethValue := st.msg.ETHValue; ethValue != nil && ethValue.Sign() != 0 {
    st.mintBVMETH(ethValue, rules)
}
snap := st.state.Snapshot()
// on failed deposit:
st.state.RevertToSnapshot(snap) // mint remains; nonce bumped
```

Tool Used

Manual Review

Recommendation

Align revm with op-geth by ensuring the deposit BVM_ETH mint persists across transaction-level failure:

- Minimal fix in `catch_error()`: include the BVM_ETH mint deltas in the returned overlay for failed deposits, mirroring the pre-snapshot persistence in op-geth (do not reapply any `eth_tx_value` transfer on failure).

```
diff --git a/crates/op-revm/src/handler.rs b/crates/op-revm/src/handler.rs
@@ fn catch_error(&self, evm: &mut Self::Evm, error: Self::Error) -> Result<...> {
```

```

-         let state = HashMap::from_iter([(caller, account)]);
+         // Persist BVM_ETH mint for failed deposit like op-geth (pre-snapshot
↪ effect).
+         let mut bvm_acc = Account::from(
+
↪ evm.ctx().db().basic(BvmEth::ADDRESS).unwrap_or_default().unwrap_or_default(),
+         );
+         let mint_u256 = U256::from(mint.unwrap_or_default());
+         if !mint_u256.is_zero() {
+             let bal_slot = BvmEth::get_balance_slot(caller);
+             let cur_bal = evm.ctx().db().storage(BvmEth::ADDRESS, bal_slot)
+                 .unwrap_or_default().unwrap_or_default().value;
+             bvm_acc.storage.insert(bal_slot,
↪ cur_bal.saturating_add(mint_u256));
+             let sup_slot = BvmEth::get_total_supply_slot();
+             let cur_sup = evm.ctx().db().storage(BvmEth::ADDRESS, sup_slot)
+                 .unwrap_or_default().unwrap_or_default().value;
+             bvm_acc.storage.insert(sup_slot,
↪ cur_sup.saturating_add(mint_u256));
+             bvm_acc.mark_touch();
+         }
+         let state = HashMap::from_iter([(caller, account), (BvmEth::ADDRESS,
↪ bvm_acc)]);

```

Alternatively, move `BvmEth::mint` to the execution prelude at a transaction “pre-snapshot” point and ensure the transaction-level failure path does not discard pre-snapshot effects.

Discussion

RealiCZ

Addressed in this pr <https://github.com/mantle-xyz/revm/pull/10>

Issue M-1: EigenDA Proxy v1.6.3 used by Mantle contains certificate field size validation bypass [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/288>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

EigenDA Proxy v1.6.3 contains a certificate field size validation issue where the disperser can increase the size of dynamic array fields to be larger than their expected size limit with verification still passing.

Vulnerability Detail

The vulnerability exists in the V1 certificate validation logic where dynamic array fields within certificates are not properly size-bounded. The disperser can craft certificates with dynamic arrays that exceed their expected size limits while still passing verification checks.

Impact

For OP Stack rollups (including Mantle):

- **Batcher Griefing:** The disperser could overload DA commitment sizes used for batcher inbox submissions on OP x EigenDA rollups to grief operations

For Arbitrum rollups (not relevant here):

- **Storage Corruption:** Break secondary storage by providing a cert that can't be regenerated perfectly when doing get queries when deriving L2 state from the sequencer inbox

Code Snippet

```
eigenda-proxy:  
  image: mantlenetworkio/eigenda-proxy:v1.6.3  
  pull_policy: always
```

Reference: <https://github.com/mantlenetworkio/networks/blob/main/docker-compose-mainnetv2-upgrade-beacon.yml#L120C1-L122C24>

When running the Docker image, the version information shows:

```
t=2025-07-10T13:50:01+0000 lvl=info msg="Starting EigenDA Proxy Server"  
↪ role=eigenda_proxy version="" date=2025-02-25--07:47:20 commit=""
```

Based on the build date (2025-02-25--07:47:20), this version is likely v1.6.3. However, when asking Mantle's team, they confirmed it.

Tool Used

Manual Review

Recommendation

Upgrade to EigenDA Proxy v1.7.0 or v1.8.2 (avoid v1.8.0 and v1.8.1 as they have high severity bugs), which implements PR #338 to enforce V1 certificate field sizes. Reference: <https://github.com/Layr-Labs/eigenda-proxy/pull/338>

Discussion

adam-xu-mantle

Acknowledged, it will be fixed in the next version

Issue M-2: Multiple bugs and security vulnerability in go-resty v2.10.0 used by Gas Oracle Service [RE-SOLVED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/289>

Summary

The Mantle network's gas oracle service uses `github.com/go-resty/resty/v2 v2.10.0`, which contains CVE-2023-45286 and multiple other bugs that could cause application crashes, authentication failures, and data handling issues.

Vulnerability Detail

CVE-2023-45286 is a race condition in go-resty v2.10.0 where `sync.Pool.Put` is called multiple times with the same `*bytes.Buffer` during request retries. This causes `sync.Pool.Get` to return a dirty buffer containing HTTP request bodies from unrelated requests, which then get concatenated and sent to potentially different servers. The vulnerability was fixed in v2.11.0.

Additionally, v2.10.0 contains several other bugs including nil pointer dereference crashes in response logging, data races in JSON logging, digest authentication failures, buffer management issues, and incorrect content type detection.

Impact

The current codebase does not explicitly enable retry functionality, so the security vulnerability does not affect the current implementation. However, if retry mechanisms are enabled in future development, HTTP request bodies could be disclosed to unintended servers.

The other bugs could cause:

- **Application Crashes:** Nil pointer dereference and race conditions can crash the gas oracle service
- **Authentication Failures:** Digest auth bugs could prevent API access to price oracles
- **Data Corruption:** Buffer management and content type detection issues could cause incorrect price data

Code Snippet

```
// usage in Client in tokenratio.go
func NewClient(url, uniswapURL string, frequency uint64) *Client {
```

```
client := resty.New() // Uses buggy v2.10.0
client.SetHostURL(url)
// No retry configuration currently - security vulnerability dormant
client.OnAfterResponse(func(c *resty.Client, r *resty.Response) error {

    })
}
```

[gas-oracle/tokenratio/tokenratio.go#L61](https://github.com/gas-oracle/tokenratio/tokenratio.go#L61)

Tool Used

Manual Review

Recommendation

Upgrade to github.com/go-resty/resty/v2 v2.16.2 or later to address all known bugs and the security vulnerability.

Discussion

pandainzoo

thanks for report, already fix <https://github.com/mantlenetworkio/mantle-v2/pull/223>

Issue M-3: Frame data loss during L2 reorg in MantleDA batch submission [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/290>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The `mantleDALoop()` function in the batch submitter fails to publish final compressed frames before clearing state during L2 reorg handling. When `Close()` is called, it flushes the compression pipeline and creates final frames, but these frames are immediately discarded by the subsequent `Clear()` call, resulting in data loss.

Vulnerability Detail

Both the regular `loop()` and `mantleDALoop()` use the same compression system where L2 blocks are compressed into frames before transmission. During normal operation, blocks are added to a compression pipeline and converted to frames via `outputFrames()`, then retrieved by `TxData()`. When an L2 reorg occurs, the current channel needs to be closed to flush any remaining compressed data.

The `Close()` method performs two critical operations:

1. Calls `s.pendingChannel.Close()` to mark the channel for termination
2. Calls `s.outputFrames()` to flush the compression pipeline and create final frames

However, in `mantleDALoop()`, after calling `Close()` to create these final frames, the code immediately calls `Clear()` which resets all state without first publishing the frames. The frames are destroyed when `Clear()` calls `clearPendingChannel()` which sets `s.pendingChannel = nil`, destroying the entire `channelBuilder` object where frames are stored in the `frames []frameData` array.

The issue occurs in this sequence:

```
err := l.state.Close()           // Creates final frames in pendingChannel.frames[]
l.state.Clear()                  // Sets pendingChannel = nil, destroying all frames
```

Impact

Data Loss: Compressed L2 block data that hasn't been converted to frames is permanently lost during reorgs and not submitted to MantleDA (EigenDA).

Code Snippet

`mantleDALoop():`

```
if errors.Is(err, ErrReorg) {
    err := l.state.Close() // Flushes compression → creates final frames
    if err != nil {
        l.log.Error("error closing the channel manager to handle a L2 reorg",
            ↳ "err", err)
    }
    l.state.Clear() // Discards final frames without publishing
    l.state.clearMantleDAStatus()
    continue
}
```

[batcher/driver_da.go#L49-L50](#)

Regular loop():

```
if err := l.loadBlocksIntoState(l.shutdownCtx); errors.Is(err, ErrReorg) {
    err := l.state.Close()
    if err != nil {
        l.log.Error("error closing the channel manager to handle a L2 reorg",
            ↳ "err", err)
    }
    l.publishStateToL1(queue, receiptsCh, true) // Publishes final frames first
    l.state.Clear()
    continue
}
```

Tool Used

Manual Review

Recommendation

Add a call to `publishStateToMantleDA()` after `Close()` and before `Clear()` to ensure final frames are published.

```
if errors.Is(err, ErrReorg) {
    err := l.state.Close()
    if err != nil {
        l.log.Error("error closing the channel manager to handle a L2 reorg",
            ↳ "err", err)
    }
    l.publishStateToMantleDA() // Publish final frames before clearing
    l.state.Clear()
    l.state.clearMantleDAStatus()
}
```

```
    continue  
}
```

This ensures that any remaining compressed data is properly flushed and submitted to MantleDA before the state is cleared, maintaining consistency with the regular loop behavior and preventing data loss during reorgs.

Discussion

adam-xu-mantle

Same as issue #297

koolexcrypto

@adam-xu-mantle not sure if it is the same. as in the other issue, no re-org should occur.

Issue M-4: tokenRatio ignored when it is 0 [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/302>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The gas calculation logic divides by `tokenRatio` and skips the division when `tokenRatio` is 0.

Vulnerability Detail

`innerExecute()` retrieves `tokenRatio` from storage and uses it to update `st.gasRemaining`:

```
tokenRatio := st.state.GetState(types.GasOracleAddr,
    ↪ types.TokenRatioSlot).Big().Uint64()
...
if tokenRatio > 0 {
    st.gasRemaining = st.gasRemaining / tokenRatio
}
```

If `tokenRatio` is 0, this division is skipped to avoid division-by-zero panic. However, if `tokenRatio` is 0, it indicates a problem elsewhere because it shouldn't be 0 if gas oracle works correctly. Hence, an error should be returned here if `tokenRatio` is 0.

Impact

Gas accounting will be incorrect if `tokenRatio` is 0 opening up the network to spam and DoS attack vectors.

Code Snippet

[state_transition.go#L722-L724](#):

```
if tokenRatio > 0 {
    st.gasRemaining = st.gasRemaining / tokenRatio
}
```

Tool Used

Manual Review

<https://github.com/mantlenetworkio/mantle-v2/blob/main/gas-oracle/tokenratio/tokenratio.go>
[allowbreak #L38](#)

lpetroulakis

Marked as acknowledged. No need to be fixed at this time.

Issue M-5: Frame reference validation error handling discrepancy [RESOLVED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/432>

Summary

The Rust and Go implementations handle empty quorum IDs in frame references differently - Go returns an error immediately, while Rust logs a warning and continues the tx loop.

Vulnerability Detail

When processing a frame reference with empty quorum IDs, Go treats empty quorum IDs as a critical error that invalidates the entire batch, while Rust treats it as a recoverable error, skipping only the problematic frame.

Impact

Correct ZK proof may not be generated to prove L2 blocks.

Code Snippet

[calldata_source.go#L369:](#)

```
if len(frameRef.QuorumIds) == 0 {  
    log.Warn("decoded frame ref contains no quorum IDs", "index", j, "err", err)  
    return nil, nil, err // stops processing entire batch  
}
```

[eigen_da.rs#L140:](#)

```
if frame_ref.quorum_ids.is_empty() {  
    warn!(target: "eigen-da-source", "decoded frame ref contains no quorum IDs");  
    continue; // skips only this frame  
}
```

Tool Used

Manual Review

Recommendation

Match the behavior across both implementations.

Discussion

RealICZ

Addressed in this pr <https://github.com/mantle-xyz/hydro/pull/1>

Issue M-6: Invalid empty check on fixed-size array [RESOLVED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/435>

Summary

A zero check is incorrectly done on the length of the array instead of on its content.

Vulnerability Detail

`blob_get()` attempts to check if a fixed-size array `[0u8; 32]` is empty using `is_empty()`, which will always return false for fixed-size arrays. The comment indicates the actual intent is to check if all bytes are zero, not if the array is empty.

The intent is to detect when field element is 0, which means all 32 bytes should be zero. However, `is_empty()` on a `[u8; 32]` checks if the array has zero length.

Impact

Code Snippet

[provider.rs#L85-L96](#):

```
let mut field_element = [0u8; 32];
self.oracle.get_exact(
    PreimageKey::new(*keccak256(blob_key), PreimageKeyType::GlobalGeneric),
    &mut field_element,
)
.await
.map_err(OracleProviderError::Preimage)?;

// if field element is 0, it means the host has identified that the data
// has breached eigenda invariant, i.e cert is valid
if field_element.is_empty() {
    return Err(OracleProviderError::Preimage(PreimageOracleError::Other(
        "field element is empty, breached eigenda invariant".into(),
    )));
}
```

Tool Used

Manual Review

Recommendation

Check if all elements the array are 0.

Discussion

RealicZ

I changed it to `if field_element != [0u8; 32]`, but later realized that since it's inside a for loop, we shouldn't be checking for emptiness there. The EigenDA team has also removed this check in their repository, see: https://github.com/Layr-Labs/hokulea/blob/master/crates/proof/src/eigenda_provider.rs [allowbreak #L141-L170](#). After testing, we confirmed that it's safe, so I removed the empty-check logic. Addressed in this pr <https://github.com/mantle-xyz/hydro/pull/2>

Issue M-7: Raw blob is used to compute KZG commitment [RESOLVED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/439>

Summary

Locally computed commitment doesn't match commitment fetched from EigenDA because raw blob is used to compute the witness instead of the encoded blob.

Vulnerability Detail

EigenDA at the time of storing a blob, encodes it and then returns a commitment to the blob. At the time of fetching this blob, it is recommended to verify that it matches the KZG commitments in the certification ([ref](#)).

Hydro tries to verify this but fails (hence, the verification code is commented) at .

Here, `last_commitment` is the locally computed commitment from the raw blob:

```
let mut witness = EigenDABlobWitness::new();

let _ = witness
    .push_witness(&blob)
    .map_err(|e| anyhow!("eigen da blob push witness error {e}"))?;

// let last_commitment = witness.commitments.last().unwrap();
```

`cert_blob_info.blob_header.commitment` is the commitment fetched from EigenDA and is a commitment of the encoded blob. This can be verified via EigenDA proxy [Put\(\)](#) function which verifies the commitment against encoded blob:

```
err = e.verifier.VerifyCommitment(cert.BlobHeader.GetCommitment(), encodedBlob)
```

Impact

Commitment verification is skipped for the data retrieved from EigenDA proxy which puts complete trust in the security of the proxy and the connection to the proxy.

Code Snippet

[handler.rs#L131-L145](#):

```

let _ = witness
    .push_witness(&blob)
    .map_err(|e| anyhow!("eigen da blob push witness error {e}"))?;

// let last_commitment = witness.commitments.last().unwrap();

// make sure locally computed proof equals to returned proof from the provider
// TODO In fact, the calculation result following the EigenLayer approach is not
// ↪ the same as the cert blob info.
// if last_commitment[..32] != cert_blob_info.blob_header.commitment.x[..]
//     || last_commitment[32..64] != cert_blob_info.blob_header.commitment.y[..]
// {
//     return Err(
//         anyhow!("proxy commitment is different from computed commitment proxy",
//     ));
// };

```

Tool Used

Manual Review

Recommendation

Use `eigenda_blob` to compute witness instead of the raw blob.

Discussion

RealicZ

Addressed in this pr <https://github.com/mantle-xyz/hydro/pull/2>

Issue M-8: Wrong revm branch imported disables mainnet env validation in op-revm [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/450>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The project imports `revm` from the `skadi_new` branch instead of the audited `dev` branch commit `59cbd48`, resulting in `op-revm`'s `validate_env()` skipping canonical mainnet validation.

Vulnerability Detail

The dependency pins in the root `Cargo.toml` in `op-succinct`, `kona` and `hydro` select `revm` and `op-revm` from `mantle-xyz/revm@skadi_new`, not the audited `dev` branch commit `59cbd48`. In `skadi_new`, `op-revm`'s `validate_env()` has the delegation to mainnet validation commented out, so after the Op-specific deposit/system transaction exceptions, ordinary transactions do not undergo the canonical non-state environment checks (e.g., intrinsic gas bounds, block/tx constraints under the active fork, tx-type/field consistency).

The expected behavior in the audited scope is represented by `dev@59cbd48`, where that delegation is present; the imported `skadi_new` diverges from it. See the referenced comparison between the imported and audited revisions: [mantle-xyz/revm compare 59cbd48..18d1633](#).

Impact

With mainnet env checks disabled for non-deposit transactions, invalid or out-of-spec parameters can pass pre-execution, leading to incorrect intrinsic gas acceptance, fork rule violations, or block/tx inconsistencies that propagate into execution. This risk compounds with other handler logic, e.g., [wei-vs-gas mis-accounting](#) when `effective_gas_price == 0` identified in a related report, which can trigger panics or incorrect gas-limit enforcement if canonical validation isn't active to gate such inputs.

Code Snippet

```
fn validate_env(&self, evm: &mut Self::Evm) -> Result<(), Self::Error> {  
    // ... Op-specific exceptions (e.g., deposit/system tx handling) ...  
  
    // self.mainnet.validate_env(evm) // commented out in imported branch
```

```
Ok()  
}
```

Tool Used

Manual Review

Recommendation

Import the audited dev branch at the exact commit 59cbd48 and pin by SHA for reproducibility.

Discussion

Realicz

We follow the code from the dev branch. The fix for `validate_env` can be found in the <https://github.com/mantle-xyz/revm/pull/10>

0xEVom

Noting here that this issue is still present in the `skadi_new` branch and is not addressed by <https://github.com/mantle-xyz/revm/pull/10>. As per the comment above, the `dev` branch will be used instead, where the issue is not present.

Issue M-9: Possible to self-transfer BVM_ETH amount greater than balance [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/460>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

A self-transfer of BVM_ETH can be done for any amount regardless of BVM_ETH balance of the sender.

Vulnerability Detail

The check that the sender has sufficient BVM_ETH balance for a transfer is done after an early return for self-transfer:

```
if from == to {  
    return Ok(());  
}  
  
let from_amount = Self::get_balance(context, from)?;  
let to_amount = Self::get_balance(context, to)?;  
  
if from_amount < eth_value {  
    return Err(OpTransactionError::BvmEth(BvmEthError::InsufficientFunds));  
}
```

This makes it possible for an address to transfer any amount of BVM_ETH to itself.

Impact

An address can send BVM_ETH amount higher than its balance to itself.

Code Snippet

[bvm_eth.rs#L203-L205](#)

Tool Used

Manual Review

Recommendation

Move the self-transfer check after the balance check.

Discussion

RealicZ

BVM_ETH does **not allow transfers to oneself**. The transfer logic of BVM_ETH is implemented by directly modifying the slot. Relevant code:
https://github.com/mantle-xyz/revm/blob/59cbd488006b8b1cd557a704b30e612c9a67445b/crates/op-revm/src/transaction/bvm_eth.rs
[allowbreak #L207-L223](#). Therefore, after this check, no funds will be moved.

This is consistent with the logic in op-geth:
https://github.com/mantlenetworkio/op-geth/blob/main/core/state_transition.go
[allowbreak #L1062-L1097](#)

Oxbok

@PinelliaC the `from == to` check is just before the code you highlighted:
https://github.com/mantle-xyz/revm/blob/59cbd488006b8b1cd557a704b30e612c9a67445b/crates/op-revm/src/transaction/bvm_eth.rs
[allowbreak #L202-L205](#)

It early returns if `from == to`.

doesn't that mean self-transfer is possible?

RealicZ

@Oxbok The actual code that changes the balance for BVM_ETH is here:
https://github.com/mantle-xyz/revm/blob/59cbd488006b8b1cd557a704b30e612c9a67445b/crates/op-revm/src/transaction/bvm_eth.rs
[allowbreak #L217-L218](#)

At this point, since `from == to`, it returns early. Doesn't that mean no operation is actually performed?

Oxbok

nonce is always incremented so an operation is performed. it's just that when `from==to`, transfer event is not emitted. Balance will not be changed because the sender and receiver address is the same.

so the code allows a self-transfer for any amount (note that this flow is triggered during a transfer for deposits): the result is a nonce increment.

Correct thing here would be to revert this operation. I suggest this diff:

```
-if from == to {  
-    return Ok(());  
-}
```

```

let from_amount = Self::get_balance(context, from)?;
let to_amount = Self::get_balance(context, to)?;

if from_amount < eth_value {
    return Err(OpTransactionError::BvmEth(BvmEthError::InsufficientFunds));
}

+if from == to {
+    return Ok(());
+}

```

You can even consider emitting an event when `from == to` since usually ERC20s emit events on self-transfer.

RealicZ

@0xbok The current self-transfer logic aligns exactly with op-geth's `transferBVMETHOD` implementation in Go ([core/state_transition.go](#)), where self-transfers early-return before balance validation, allowing arbitrary `ethValue` as a noop (nonce++ only, no balance change or event).

0xEVom

@RealicZ the behaviour here may be consistent with op-geth, but it is not the expected behaviour and can have unexpected side effects. I warmly recommend you follow @0xbok's advice and update both revm and geth.

RealicZ

@0xbok @0xEVom We discussed this internally, and we'll apply the corresponding updates to both revm and geth in the next release. Thanks for the thorough manual review!

Issue M-10: Safe L1 head selection uses a time-dependent offset, breaking determinism and DA anchoring [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/461>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The EigenDA host computes the safe L1 head by adding a hardcoded offset and clamping to finalized instead of selecting the L1 block that actually includes the EigenDA-referenced batches for the target L2 span, making the proving anchor time-dependent and misaligned with DA inclusion.

Vulnerability Detail

In `utils/eigenda/host/src/host.rs`, `calculate_safe_l1_head()` calls `fetcher.get_l1_head()` to derive an L1 height suitable for the requested L2 range, then unconditionally adds +20 blocks before clamping to `fetcher.get_l1_header(BlockId::finalized())`. The function comment states it should “find the latest L1 block containing batches with EigenDA data,” but the implementation introduces a heuristic offset carried over from the Ethereum DA path, which is not needed under EigenDA with a DA-aware op-node and breaks DA anchoring.

This leads to:

- Time-dependent anchors: the finalized cap advances with time; adding a fixed offset yields different `l1_head` values for the same L2 span when the job is re-run at different times. Since `l1_head` is part of the prover's boot args and determines the L1 headers embedded in the witness, proofs from separate runs are non-interchangeable.
- DA decoupling: the selected L1 head may be inclusion+20, and under a fallback mode could even predate inclusion, causing data retrieval retries. The anchor thus fails to encode the intended boundary, i.e., the latest L1 block that actually includes EigenDA-referenced batches.

By contrast, the Celestia host computes the safe head from Blobstream commitments, deterministically matching DA inclusion.

Impact

The system proves against inconsistent L1 boundaries that are not anchored to DA inclusion:

- Re-running the same L2 span at different times produces different `l1_head` values, changing the L1 headers embedded in the witness; prior witnesses cannot be reused byte-for-byte.
- If `SAFE_DB_FALLBACK==true`, anchors may not include the required EigenDA references.

Code Snippet

```
// utils/eigenda/host/src/host.rs
async fn calculate_safe_l1_head(
    &self,
    fetcher: &OPSuccinctDataFetcher,
    l2_end_block: u64,
    safe_db_fallback: bool,
) -> Result<B256> {
    // For Ethereum DA, use a simple approach with minimal offset.
    let (_, l1_head_number) = fetcher.get_l1_head(l2_end_block,
        ↪ safe_db_fallback).await?;

    // FIXME: Investigate requirement for L1 head offset beyond batch posting block
    // Add a small buffer for Ethereum DA.
    let l1_head_number = l1_head_number + 20;

    // Ensure we don't exceed the finalized L1 header.
    let finalized_l1_header = fetcher.get_l1_header(BlockId::finalized()).await?;
    let safe_l1_head_number = std::cmp::min(l1_head_number,
        ↪ finalized_l1_header.number);

    Ok(fetcher.get_l1_header(safe_l1_head_number.into()).await?.hash_slow())
}
```

Tool Used

Manual Review

Recommendation

- Anchor the safe L1 head to the L1 block returned by `get_l1_head()` (EigenDA-aware via `op-node`) and remove both the hardcoded offset and the redundant clamp. The clamp is unnecessary because `get_l1_head()` already bounds the result to L1 finalized.
- Ensure `SAFE_DB_FALLBACK=false` in production.

Discussion

RealicZ

Celestia retrieves the corresponding L1 address through contract events: https://github.com/succinctlabs/op-succinct/blob/main/utils/celestia/host/src/blobstream_utils.rs#L98-L184

However, the `batch_inbox_address` of eigenDA-V1 is an EOA address: <https://sepolia.etherscan.io/address/0xffeeddccbbaa00>

Therefore, it is not possible to obtain the L1 block in the same way.

Currently, we believe there is no risk, as the method of fetching the L1 block has been tested extensively over a long period without issues. We will reevaluate this once EigenDA-V2 is introduced.

0xEVom

Thanks, agree that event-based anchoring (Celestia) is not applicable with an EOA inbox on EigenDA v1. The point here isn't to use events, but to rely on the DA-aware anchor you already expose via op-node:

- op-node already records the L1 origin that makes a given safe L2 head derivable and serves it via `optimism_safeHeadAtL1Block` (SafeDB). In your fork, as far as I understand it SafeDB is updated by the derivation pipeline when safe head advances, and that pipeline derives from EigenDA; so `get_l1_head()` is the right DA-aware source of truth.
- Given that, the +20 offset in the EigenDA host is unnecessary and introduces time-dependence into the proving anchor. The finalized clamp then becomes redundant and can also be dropped.

In practice this is low-risk and if it remains stable, it's absolutely fine to revisit when EigenDA v2 lands.

Issue M-11: KZG blob proofs are not strictly verified in client blob provider [RESOLVED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/464>

Summary

`verify_blob_kzg_proof_batch()` returns `Result<bool, KzgError>` but the boolean result is not checked; only the error case is handled. This could accept invalid EIP-4844 blobs when the batcher falls back to posting blob transactions.

Vulnerability Detail

In `BlobStore::from()` in `utils/client/src/oracle/blob_provider.rs`, the client constructs `kzg_rs::Blobs` and calls `kzg_rs::KzgProof::verify_blob_kzg_proof_batch()`. That function returns `Result<bool, KzgError>`, where `Ok(true)` means the proofs are valid and `Ok(false)` means they are invalid. The call site uses `.expect(...)` on the `Result`, which only fails on `Err(e)` and does not require `Ok(true)`. Consequently, `Ok(false)` is treated as success and invalid blobs would be accepted.

Although derivation is currently wired to EigenDA only, the OP Stack batcher used in this stack explicitly falls back to EIP-4844 blob transactions when EigenDA dispersal is skipped or fails. See [op-batcher/batcher/driver_da.go](#): when EigenDA dispersal does not succeed, `blobTxCandidates(...)` are created and submitted. The fallback can also be forced via the CLI flag `--skip-eigenda-da-rpc` (`SkipEigenDaRpc`) defined in [op-batcher/flags/flags.go](#). In either case, the 4844 path in the client becomes live and this verification gap becomes reachable.

This issue was reported upstream and disclosed in this [security advisory](#), with a [fix](#) merged to require `Ok(true)` explicitly.

Impact

When the batcher posts EIP-4844 blob transactions (fallback or by configuration), invalid blobs whose KZG verification returns `Ok(false)` can be accepted. This allows derivation over tampered data, producing an incorrect state transition.

Code Snippet

```
// utils/client/src/oracle/blob_provider.rs
kzg_rs::KzgProof::verify_blob_kzg_proof_batch(
    blobs,
    value.commitments,
    value.proofs,
```

```

    &get_kzg_settings(),
)
.expect("Failed to verify blob KZG proofs.");

```

Tool Used

Manual Review

Recommendation

Adopt the upstream fix pattern and require `Ok(true)` explicitly, rejecting `Ok(false)` and surfacing `Err(e)` clearly.

```

diff --git a/utils/client/src/oracle/blob_provider.rs
↪ b/utils/client/src/oracle/blob_provider.rs
@@ -25,11 +25,17 @@ impl From<BlobData> for BlobStore {
-     kzg_rs::KzgProof::verify_blob_kzg_proof_batch(
-         blobs,
-         value.commitments,
-         value.proofs,
-         &get_kzg_settings(),
-     )
-     .expect("Failed to verify blob KZG proofs.");
+     match kzg_rs::KzgProof::verify_blob_kzg_proof_batch(
+         blobs,
+         value.commitments,
+         value.proofs,
+         &get_kzg_settings(),
+     ) {
+         Ok(true) => {} // verified
+         Ok(false) => panic!("KZG proof verification failed: invalid proofs"),
+         Err(e) => panic!("KZG proof verification error: {e}"),
+     }

```

Discussion

byteflyfunny

will be fixed

0xEVom

Fixed in <https://github.com/mantle-xyz/op-succinct/pull/24>

Issue M-12: Consensus gas delta on deposits due to fixed 4500 gas subtraction [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/474>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

On deposits that include `ETHValue` and a non-empty `input`, `op-revm` subtracts a fixed 4500 gas in `refund()`. `op-gets` has no such adjustment in its gas/refund path. This deterministically changes `UsedGas`/refunds and can cause execution/receipt mismatches.

Vulnerability Detail

`op-revm` introduces a fixed subtraction in `refund()` that applies when `tx.eth_value()` is set and `tx.input()` is non-empty. The code executes before the final refund capping and token-ratio scaling, pushing a 4500 gas delta into final accounting. See `refund()` in `crates/op-revm/src/handler.rs` at [L384-L388](#).

In `op-gets`, runtime gas accounting and refund handling do not include any fixed 4500 offset. For non-deposits, `11Gas` is subtracted and remaining gas is divided by `tokenRatio` at [core/state_transition.go#L715-L724](#). Refund computation and scaling occur at [#L803-L810](#), with capping implemented by `calcRefund()` at [#L947-L964](#). For deposits under `Regolith`, `gets` finalizes execution at [#L833-L839](#) without any fixed subtraction. There is no conditional on `ETHValue`/`input` that subtracts 4500 gas in these locations.

The absence of a fixed deduction in `op-gets` and its presence in `op-revm` means the two clients will disagree on `gas.remaining` just before final refund and scaling, which propagates into `UsedGas` and refunds.

With Mantle [configured](#) with `Regolith` active from genesis, this difference is live.

Impact

Deposits with `ETHValue` and non-empty `input` will have `UsedGas` increased by 4500 and refunds reduced by 4500 units in `op-revm` relative to `op-gets`. Execution/receipt mismatches can cause ZK proof derivations to disagree with the sequencer's executed state.

Code Snippet

```
// crates/op-revm/src/handler.rs
fn refund(&self, evm: &mut Self::Evm, exec_result: &mut <Self::Frame as
↳ Frame>::FrameResult, eip7702_refund: i64) {
    ...
    let is_eth_mint = tx.eth_value().is_some();
    if is_eth_mint && !tx.input().is_empty() {
        gas.set_remaining(gas.remaining().saturating_sub(4500)); // fixed
        ↳ deduction; no geth counterpart
    }
    ...
}
```

Tool Used

Manual Review, Cursor

Recommendation

Align behaviour between op-revm and op-geth. If there is no protocol requirement for a fixed deduction, remove it in op-revm. If intentional, implement the same logic at the equivalent stage in op-geth.

Discussion

RealiCZ

This behavior is known – when `is_eth_mint` is set and `tx.input()` is non-empty, the gas usage differs by a consistent 4500 units between op-revm and op-geth. We've verified through extensive testing that the delta is always 4500, though the underlying cause is still under investigation. As a temporary workaround, subtracting this 4500 magic number ensures execution consistency. A proper fix will be included in the next release.

Issue L-1: Pending issues from previous audits [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/285>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

This issue is to re-report findings from previous audits that are not fixed yet.

Vulnerability Detail

Sigma Prime's Mantle EigenDA Integration audit:

- MEDA-18 `EigenDaClient.DisperseBlob()` Does Not Handle HTTP 503 Error

OpenZeppelin's Mantle Op-geth Op-stack Diff Audit

- L-04 Direct Communication with EigenDA Disperser in `RetrieveBlob`

Impact

Depends on the individual issues.

Tool Used

Manual Review

Recommendation

Fixed the open issues from previous audits.

Discussion

adam-xu-mantle

Acknowledged, it will be fixed in the next version

Issue L-2: Some op-batcher config parameters can be removed [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/292>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Some config parameters are not in use anymore due to the move to eigenDA.

Vulnerability Detail

Some config parameters related to Mantle DA are defined but not used. These parameters can be removed.

Impact

Redundant code.

Code Snippet

[config.go#L50-L59:](#)

```
DisperserSocket          string
DisperserTimeout         time.Duration
DataStoreDuration        uint64
GraphPollingDuration     time.Duration
RollupMaxSize            uint64
MantleDaNodes            int
DataLayrServiceManagerAddr common.Address
DataLayrServiceManagerContract *bindings.ContractDataLayrServiceManager
DataLayrServiceManagerABI  *abi.ABI
```

Tool Used

Manual Review

Recommendation

Remove the params highlighted above.

Discussion

adam-xu-mantle

Acknowledged, it will be fixed in the next version

Issue L-3: Empty TLS config to connect to eigenDA proxy [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/294>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

`RetrieveBlob()` and `GetBlobStatus()` uses empty TLS config to connect to eigenDA proxy.

Vulnerability Detail

An empty TLS config typically results in insecure default settings that accept any certificate, including self-signed or invalid ones, making the connection vulnerable to man-in-the-middle (MITM) attacks.

Impact

The lack of proper TLS configuration allows attackers to intercept, modify, or inject malicious data during blob retrieval operations.

Code Snippet

[da_proxy.go#L67](#), [da_proxy.go#L178](#):

```
config := &tls.Config{}
```

Tool Used

Manual Review

Recommendation

Enforce secure TLS configuration.

Discussion

adam-xu-mantle

Acknowledged

Issue L-4: Anyone can submit blobs if they know the proxy URL [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/295>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

DisperseBlob() isn't adding any auth header to the HTTP request which indicates eigenDA proxy server is accepting requests from anyone.

Vulnerability Detail

DisperseBlob() isn't setting any auth header to the HTTP request. This call is made to disperse blobs in EigenDA network. This lack of auth header suggests that the eigenDA proxy server run by Mantle isn't authenticating the caller, hence anyone can submit blobs if they know the proxy URL.

Impact

Attackers can spam blob submissions to consume storage/bandwidth of the proxy server

- Could lead to DoS against legitimate users
- Potential financial impact from EigenDA network fees

Code Snippet

[da_proxy.go#L147](#):

```
req, err := http.NewRequestWithContext(ctx, http.MethodPost, url, body)
req.Header.Set("Content-Type", "application/octet-stream") // @audit -> no auth
↪ headers
```

Tool Used

Manual Review

Recommendation

Add authentication mechanism to eigenDA proxy.

Discussion

adam-xu-mantle

Acknowledged

Issue L-5: Integer overflow in SafeHeadAtL1() when querying maximum L1 block number [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/298>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Integer overflow vulnerability in SafeHeadAtL1() when querying for the maximum possible L1 block number due to unchecked arithmetic operation.

Vulnerability Detail

The SafeHeadAtL1() function performs unchecked addition on a uint64 parameter:

```
if valid := iter.SeekLT(safeByL1BlockNumKey.Of(l1BlockNum + 1)); !valid {  
    err = ErrNotFound  
    return  
}
```

When l1BlockNum equals math.MaxUint64 (18,446,744,073,709,551,615), the expression l1BlockNum + 1 wraps around to 0 due to integer overflow. This causes SeekLT to search for entries before block 0, which will always fail since block 0 represents the smallest possible key in the database.

The intended logic is to find the largest L1 block number \leq l1BlockNum by seeking the largest key less than l1BlockNum + 1. However, the overflow breaks this logic for the edge case of maximum uint64 values.

Impact

This is an edge case which may not be realized in production, however if the max value is supposed to be a placeholder to find the latest safe head, then the impact is more severe. The function will incorrectly return ErrNotFound when querying for safe head information at the maximum possible L1 block number.

Code Snippet

[safedb.go#L175](#):

```
if valid := iter.SeekLT(safeByL1BlockNumKey.Of(l1BlockNum + 1)); !valid {  
    err = ErrNotFound
```

```
    return  
}
```

Tool Used

Manual Review

Recommendation

Add an overflow check before the arithmetic operation.

Discussion

pandainzoo

thanks for report, currently, there is no impact as the L1 block number will not overflow. We plan to fix it in the next release.

Issue L-6: SafeHeadReset() doesn't clear batch if an error occurs [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/299>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Missing resource cleanup for database batch causes potential memory leak in SafeHeadReset().

Vulnerability Detail

The SafeHeadReset() function creates a database batch but fails to properly clean up the resource. This is unlike SafeHeadUpdated() in the same file which correctly uses defer batch.Close().

Impact

Memory consumption by uncommitted batches.

Code Snippet

[safedb.go#L142](#):

```
batch := d.db.NewBatch()
```

Tool Used

Manual Review

Recommendation

Add proper resource cleanup:

```
batch := d.db.NewBatch()  
+defer batch.Close()
```

Discussion

pandainzoo

thanks for report, we will fix this issue in the next version.

Issue L-7: Mantle DA switch not implemented in Kona [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/429>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Mantle's Kona doesn't support the DA switch to toggle the DA layer between EigenDA and EOA calldata.

Vulnerability Detail

op-batcher and op-node depending on the configured `mandleDaSwitch` uses EigenDA (with Ethereum blobs as fallback) or EOA calldata as the DA layer.

However, kona doesn't support this switch currently and only uses EigenDA (with Ethereum blobs as fallback) to read data. This is also documented in a TODO comment.

Impact

If the switch is turned off for op-batcher and op-node, the related ZK proofs won't be generated.

Code Snippet

[ethereum.rs#L39](#):

```
// [TODO]: add cfg.mantle_da_switch !!
```

Tool Used

Manual Review

Recommendation

Add support for DA switch in Kona.

Discussion

RealICZ

The ZKP doesn't need to process the previous `mantle_da` data, and we **won't be switching back** to the non-EigenDA submission method going forward, so no extra compatibility is needed.

lpetroulakis

Acknowledged by the team but no remediation needed.

Issue L-8: Blob index tracking discrepancy between Go and Rust implementations [RESOLVED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/430>

Summary

The Rust implementation of `data_from_eigen_da()` fails to increment the blob index counter when encountering transactions with no recipient address (`to` is `None`), while the Go implementation increments the blob index by the number of blob hashes. This discrepancy causes blob index misalignment between different implementations.

Vulnerability Detail

In the Go implementation, when `isValidBatchTx()` returns false (including when `to` is nil), the blob index is incremented:

```
if !isValidBatchTx(tx, config.L1Signer(), config.BatchInboxAddress, batcherAddr) {
    blobIndex += len(tx.BlobHashes()) // Always increments for invalid txs
    continue
}
```

However, in the Rust implementation, when `tx_kind` (the `to` field) is `None`, the code simply continues without incrementing the blob counter:

```
let Some(to) = tx_kind else { continue }; // No blob counter increment!
if to != self.batcher_address {
    number += blob_hashes.map_or(0, |h| h.len() as u64);
    continue;
}
```

This means that for transactions without a recipient address that contain blobs, the Rust version will have a lower blob index than the Go version, causing the two implementations to diverge in their blob indexing.

Impact

Correct ZK proof may not be generated to prove L2 blocks, although if everything works correctly, a blob tx should never have a null `to` address

Code Snippet

[eigen_da.rs#L104](#), [calldata_source.go#L417-L419](#):

```
// Rust - Missing blob index increment
let Some(to) = tx_kind else { continue }; // Bug: Should increment number before
↳ continue

// Go - Correctly increments blob index
if to == nil || *to != batchInboxAddr {
    return false // Caller increments blobIndex
}
```

Tool Used

Manual Review

Recommendation

Update the Rust implementation to increment the blob counter before continuing when `tx_kind` is `None`.

Discussion

Realicz

Addressed in this pr <https://github.com/mantle-xyz/hydro/pull/1>

OxEVom

Adjusting severity to low since as noted under Impact, EIP-4844 txs do not accept a nil to field.

Issue L-9: CalldataFrame processing logic discrepancy between Go and Rust implementations [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/431>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The Rust implementation only processes CalldataFrame data when the first byte equals DERIVATION_VERSION_EIGEN_DA, while the Go implementation processes CalldataFrame for all non-empty calldata regardless of the first byte.

Vulnerability Detail

The control flow differs significantly between implementations.

The key difference is that Go attempts to decode all non-empty calldata as CalldataFrame, while Rust only processes calldata that starts with the EigenDA version byte.

Impact

Valid calldata without the EigenDA version prefix will be silently ignored by the Rust implementation but processed by Go. This can be realized when Mantle DA switch is turned on or there is some unknown issue with data encoding.

Code Snippet

[calldata_source.go#L351-L401](#):

```
switch {
case len(data) == 0:
    // Handle empty data
    continue
case data[0] == eigenda.DerivationVersionEigenDa:
    // Strip version byte
    data = data[1:]
}
// Process ALL non-empty data as CalldataFrame
calldataFrame := &op_service.CalldataFrame{}
err := proto.Unmarshal(data, calldataFrame)
```

[eigen_da.rs#L130-L154](#)::

```
if calldata.len() == 0 {
    // Handle empty data
    continue;
}
if calldata[0] == DERIVATION_VERSION_EIGEN_DA {
    // ONLY process if version byte matches
    let blob_data = calldata.slice(1..);
    let calldata_frame: CalldataFrame = CalldataFrame::decode(blob_data)?;
    // ... rest of processing
}
// Implicitly: non-EIGEN_DA calldata is skipped
```

Tool Used

Manual Review

Recommendation

Update op-node to process CallDataFrames only when the calldata version matches EigenDA derivation version.

Discussion

RealicZ

same as #429—we won't easily switch back to the non-EigenDA submission method in the future

Issue L-10: Potential out-of-bounds panic in data_from_eigen_da() [RESOLVED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/434>

Summary

There are some potential out-of-bounds errors causing the Rust version of the node to panic.

Vulnerability Detail

If `frame_ref.blob_length` is greater than `blob_data.len()`, this will panic with an out-of-bounds error. There's no validation that the `blob_length` from the protobuf is valid. [eigenda.rs#L156](#):

```
let blobs = &blob_data[..frame_ref.blob_length as usize];
```

Impact

Due to unknown issues with the code, lack of validation can lead to panic in the Rust implementation of the node.

Code Snippet

```
let blobs = &blob_data[..frame_ref.blob_length as usize];
```

Tool Used

Manual Review

Recommendation

Consider doing array length checks before slicing or accessing an index.

Discussion

RealiCZ

Addressed in this pr <https://github.com/mantle-xyz/hydro/pull/1>

Issue L-11: Incorrect comment for indices storing rollup data size [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/436>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Comment says indices 2 to 4 store length of the input rollup data, but instead it is indices 2 to 5.

Vulnerability Detail

The comment at [eigenda_data.rs#L50](#) says indices 2 to 4 store rollup data length, but the code at [eigenda_data.rs#L74](#) uses indices 2 to 5 for the length.

Impact

No impact, only the comments need to correctly reflect the code.

Code Snippet

Comment:

```
/// The 2-4 indices of header are storing the length of the input rollup data in  
↪ big endian
```

Code:

```
raw_blob[2..6].copy_from_slice(&rollup_data_size.to_be_bytes());
```

Tool Used

Manual Review

Recommendation

Update the comment.

Discussion

Realicz

Acknowledged, this comment does not exist in hydro https://github.com/mantle-xyz/hydro/blob/main/crates/eigenda/src/common/eigenda_data.rs
[allowbreak #L74](#)

Issue L-12: Missing validation in EigenDABlobData::decode() [RESOLVED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/437>

Summary

The blob decode function fails to validate header fields including the version byte and required zero values, potentially allowing incorrectly formatted or incompatible blob versions to be processed.

Vulnerability Detail

The encode function sets specific header values that the decode function never validates:

```
// Encode sets these header constraints:
raw_blob[0] = 0; // Must be 0 for bn254 field element constraint
raw_blob[1] = BLOB_ENCODING_VERSION_0; // Version byte
raw_blob[2..6] = content_size; // Content length
raw_blob[6..32] = 0; // Should be zeros

// Decode only reads content size, ignoring other validations:
let content_size = blob.slice(2..6).get_u32();
let codec_data = blob.slice(32..); // Skips entire header without validation
```

Missing validations include:

Zero constraint (index 0): No verification of the bn254 field element requirement
Version byte (index 1): No check that the blob uses the expected encoding version
Reserved bytes (6-31): No validation that unused header bytes are zero
Header integrity: No overall header validation before processing

Impact

- Version Incompatibility: Future blob versions with different encoding schemes could be incorrectly processed using the current decoder
- Invalid Data Processing: Malformed blobs that don't meet the bn254 field requirements could cause cryptographic operations to fail
- Data Corruption: Processing blobs with unexpected header formats could result in corrupted output data

Code Snippet

eigenda_data.rs#L23-L33:

```
pub fn decode(&self) -> Result<Bytes, BlobDecodingError> {
    let blob = &self.blob;
    if blob.len() < 32 {
        return Err(BlobDecodingError::InvalidLength);
    }

    ...
    let content_size = blob.slice(2..6).get_u32();
    let codec_data = blob.slice(32..);
    // @audit Proceeds without validating header...
}
```

Tool Used

Manual Review

Recommendation

Discussion

RealiCZ

Addressed in this pr <https://github.com/mantle-xyz/hydro/pull/1>

Issue L-13: Several tests are not compiling in kona [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/440>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Tests aren't compiling.

Vulnerability Detail

Several tests are failing because of unresolved import or no-such-field error.

Impact

Tests are an important part of the security posture of a project. Failing tests will not be able to uncover bugs in the code.

Code Snippet

Multiple tests are failing, here is one instance: [stateful.rs#L474](#):

```
let cfg = Arc::new(RollupConfig {  
    block_time,  
    hardforks: HardForkConfig { canyon_time: Some(0), ..Default::default() }, //  
    ↪ @audit there is no such field  
    ..Default::default()  
});
```

Tool Used

Manual Review

Recommendation

Fix all the tests failing to compile.

Discussion

RealICZ

Acknowledged, it will be fixed in the next version

Issue L-14: Memory over-allocation in `frame_data()` [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/441>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

`frame_data()` allocates memory for frame overhead for each frame which isn't necessary.

Vulnerability Detail

`frame_data()` allocates `self.size()` memory for data:

```
let mut data = Vec::with_capacity(self.size());
(0..=self.last_frame_number).try_for_each(|i| {
    let frame = self.inputs.get(&i)?;
    data.extend_from_slice(&frame.data);
    Some(())
})?;
```

`self.size()` is a cumulation of `frame.sizes` which is defined as ([frame.rs#L167-L169](#)):

```
pub fn size(&self) -> usize {
    self.data.len() + FRAME_OVERHEAD
}
```

This means the memory allocated to data includes the overhead of `FRAME_OVERHEAD` bytes per frame. The actual data concatenation doesn't include this overhead, thus over-allocating 200B of memory for each frame.

Impact

Over-allocation of memory.

Code Snippet

[channel.rs#L186-L191](#),

Tool Used

Manual Review

Recommendation

Adjust capacity of data vector to remove the overhead.

Discussion

RealICZ

Acknowledged, it will be fixed in the next version

Issue L-15: Frame sequences accept duplicates and out-of-order elements without validation [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/442>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The EigenDA frame processing logic appends decoded frames without validating sequence integrity, allowing duplicate or reordered frames.

Vulnerability Detail

When processing frames from RLP-decoded blob data, the code iterates through the decoded frame list and appends each frame directly to the `data` vector without validation. This occurs in `data_from_eigen_da()` when handling `FrameRef` responses and in `load_blobs()` when processing EIP-4844 blob data.

The `FrameRef` structure contains a `blob_index` field that could be used for ordering validation, but this field is not utilised. Frames are processed in the order they appear in the RLP-decoded list, which is controlled by the external `EigenDA proxy service`. The downstream `next_data()` method uses simple FIFO processing with `data.remove(0)` without verifying sequence integrity.

Impact

Compromised or accidental provider responses could introduce duplicate frames or incorrect ordering, potentially causing logical inconsistencies in the derivation pipeline if consumers assume sequence integrity.

Code Snippet

```
// In data_from_eigen_da() - FrameRef processing
for blob in blob_data.0 {
    data.push(Bytes::from(blob));
}

// In load_blobs() - EIP-4844 blob processing
for blob in rlp_blob.0 {
    blob_data.push(Bytes::from(blob));
}
```

Tool Used

Manual Review

Recommendation

Implement frame sequence validation using the available `blob_index` field to ensure frames are processed in the correct order. Add checks to reject duplicate frames and verify monotonic ordering of indices.

Discussion

byteflyfunny

This part of the logic is aligned with Mantle v2's Golang code. For now, we plan to handle it this way, and will unify the processing once Eigen v2 is launched.

Issue L-16: Infinite retry loop without backoff [AC-KNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/443>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Temporary errors retry immediately in a tight loop without any delay or backoff.

Vulnerability Detail

`execute()` immediately retries the task on a temporary error:

```
while let Err(e) = self.execute_inner(state).await {
    match e {
        EngineTaskError::Temporary(e) => {
            trace!(target: "engine", "{e}");
            continue; // Immediate retry, no backoff
        }
    }
}
```

This can potentially overwhelm the services.

Impact

Immediate retries can lead to overwhelming the execution engine with rapid-fire requests and resource exhaustion.

Code Snippet

[task.rs#L81-L86](#)

Tool Used

Manual Review

Recommendation

Implement a delay or backoff mechanism for retries.

Discussion

byteflyfunny

This part of the logic is kept consistent with the OP Stack, and will not be modified.

Issue L-17: EigenDA KZG settings mismatch can cause inconsistent blob verification [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/445>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

EigenDA paths use different KZG backends without an explicit assertion that they share the same SRS/domain and versioned-hash semantics, risking divergent verification outcomes.

Vulnerability Detail

In the client, `blob_provider.rs` computes versioned hashes with `kzg_to_versioned_hash()` and verifies batches via `KzgProof::verify_blob_kzg_proof_batch()` using `get_kzg_settings()`. Host-side witness generation computes commitments and proofs using `EnvKzgSettings::default()` in `online_blob_store.rs`.

There is no assertion that:

- The SRS/domain used by the host (`EnvKzgSettings::default()`), the client (`get_kzg_settings()`), and Hydro's EigenDA utilities are identical.
- The client's `kzg_to_versioned_hash()` matches a reference implementation (see `CalcBlobHashV1` in `blob.go`).

Impact

Differences in SRS/domain or hashing semantics can cause valid EigenDA blobs to be rejected by the client or create mismatched versioned-hash/commitment associations.

Code Snippet

```
/* utils/client/src/oracle/blob_provider.rs */
use alloy_eips::eip4844::kzg_to_versioned_hash;
use kzg_rs::get_kzg_settings;

...
.map(|c| kzg_to_versioned_hash(c.as_slice()))
...
kzg_rs::KzgProof::verify_blob_kzg_proof_batch(
    blobs,
    value.commitments,
```

```
    value.proofs,  
    &get_kzg_settings(),  
)
```

```
/* utils/host/src/witness_generation/online_blob_store.rs */  
use alloy_eips::eip4844::env_settings::EnvKzgSettings;  
  
let settings = EnvKzgSettings::default();  
let commitment = settings.get().blob_to_kzg_commitment(&c_kzg_blob).unwrap();  
let proof = settings.get().compute_blob_kzg_proof(&c_kzg_blob,  
↳ &commitment.to_bytes()).unwrap();
```

```
/* contracts/lib/mantle-v2/op-service/eigenda/da_proxy.go */  
decodedData := RemoveEmptyByteFromPaddedBytes(reply.Data) // decode modulo bn254
```

Tool Used

Manual Review

Recommendation

- Use a single backend for proving and verifying or ensure all components load the identical SRS/domain from a single configuration source.
- Add a cross-backend consistency test asserting that commitments/proofs from `EnvKzgSettings::default()` verify under `get_kzg_settings()` and that `kzg_to_versioned_hash()` equals a reference sha256+version implementation; pin library versions; fail fast on startup if the check fails.

Discussion

byteflyfunny

Good question. EigenLayer needs to provide such a method, similar to Ethereum's KZG library. However, at the moment, projects have to maintain the KZG point version information themselves, while Eigen DA will also maintain it.

Issue L-18: EigenDA v1 commitment envelope is not validated in Hydro before decoding [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/446>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Hydro accepts EigenDA commitments without asserting the proxy-defined “mode” metadata header/version prior to decoding. This permits malformed or legacy commitment envelopes to pass into derivation.

Vulnerability Detail

The rollup-facing commitment envelope is produced by `eigenda-proxy`, which prefixes a few-byte “commitment schema” header followed by the EigenDA certificate payload. Mantle’s op-service uses `eigenda_cert_v1` per da-proxy.go. Consumers must parse and enforce this header/version to ensure only valid envelopes are accepted.

In Hydro, the commitment header/version is not asserted before decoding. Specifically:

- `EigenDAChainHintHandler::fetch_hint()` decodes `BlobInfo` from a sliced `commitment` but does not parse or enforce the `eigenda_cert_v1` header/version before decoding.
- `OracleEigenDaProvider::blob_get()` follows the same pattern, slicing and decoding without verifying the envelope header/version.

Absent a local assertion, malformed or legacy envelopes may be accepted.

Impact

Without enforcing the `eigenda_cert_v1` envelope header/version locally, Hydro can ingest malformed or legacy envelopes and proceed to derive state from data referenced by those envelopes.

Code Snippet

```
// bin/host/src/eigenda/handler.rs: decodes without header/version assertion
let cert_blob_info = BlobInfo::decode(&mut &commitment[3..])?;

// crates/oracle/src/provider.rs: decodes without header/version assertion
let cert_blob_info = BlobInfo::decode(&mut &commitment[3..]).unwrap();
```

Tool Used

Manual Review

Recommendation

Enforce `eigenda_cert_v1` envelope header/version prior to decoding:

- In `EigenDAChainHintHandler::fetch_hint()` and `OracleEigenDaProvider::blob_get()`, parse the proxy header according to the deployed `eigenda-proxy v1` specification, assert the exact bytes, and reject on mismatch. Fix the misleading comment at [handler.rs#L62](#) to reflect the correct header bytes and slice length.
- Maintain this validation even if proxy-side certificate verification is later enabled; it serves as defense-in-depth.

Discussion

byteflyfunny

This part of the code will be modified in EigenDA v2, where all verification methods may change. For the current version, no modifications will be made. The EigenDA Proxy is a trusted self-operating service, so the issue can be avoided.

lpetroulakis

Acknowledged by the team without immediate remediation needed.

Issue L-19: Panics and gas mis-accounting from unchecked `token_ratio` conversions [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/447>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Unchecked `token_ratio` conversions and arithmetic (including `unwrap()`, division, and `assert!`) can panic or distort gas/refund accounting when the value is zero or out of range. Although `token_ratio` is expected to be non-zero, the current code assumes this without defensive checks.

Vulnerability Detail

The `token_ratio` is applied at several stages in the non-deposit transaction flow:

- In `validate_initial_tx_gas()`, `initial_gas` and `floor_gas` are multiplied by `token_ratio` using `token_ratio.try_into().unwrap()` and `checked_mul`. Any non-convertible or extreme `token_ratio` will trigger a panic via `unwrap()`, and multiplication can still fail for large values.
- In `execution()` and `inspect_execution()`, the computed `gas_limit` is reduced by the LI cost and then divided by `token_ratio.try_into().unwrap()`. A zero `token_ratio` (e.g., due to stale/corrupt reads) will cause division-by-zero or a panic via `unwrap()`.
- In `refund()`, an `assert!(token_ratio_u64 <= i64::MAX)` can abort execution if the value is too large; refunds and remaining gas are then scaled by `token_ratio`, and `limit` is divided by it if non-deposit and non-system. This mixes multiple scaling points and introduces another zero/division hazard.

The value is sourced from the chain context via `get_token_ratio()` which returns `self.token_ratio.unwrap_or(U256::from(1))`. This avoids `None` but does not prevent zero or out-of-range values being used later. The combination of unchecked conversions, integer assertions, and arithmetic on a dynamic parameter can therefore cause panics or invalid gas/refund mutations. Even if `token_ratio` is never expected to be zero, stale reads or misconfiguration would propagate through these paths and crash the node.

Impact

A zero or extreme `token_ratio` can cause the client to panic at runtime (e.g., `unwrap()` or `assert!`), or corrupt gas/refund accounting by performing invalid scaling and divisions.

Code Snippet

```
/* crates/op-revm/src/handler.rs */

initial_gas.initial_gas = initial_gas
    .initial_gas
    .checked_mul(token_ratio.try_into().unwrap())
    .ok_or(InvalidTransaction::CallerGasLimitMoreThanBlock)?;

initial_gas.floor_gas = initial_gas
    .floor_gas
    .checked_mul(token_ratio.try_into().unwrap())
    .ok_or(InvalidTransaction::CallerGasLimitMoreThanBlock)?;

/* ... in execution()/inspect_execution() ... */

gas_limit = gas_limit.wrapping_sub(tx_l1_cost.try_into().unwrap());
gas_limit = gas_limit.wrapping_div(token_ratio.try_into().unwrap());

/* ... in refund() ... */

let token_ratio_u64: u64 = ctx.chain().get_token_ratio().try_into().unwrap();
assert!(
    token_ratio_u64 <= i64::MAX as u64,
    "token_ratio {token_ratio_u64} exceeds i64::MAX"
);
if !is_system && !is_deposit {
    if token_ratio_u64 > 0 {
        gas.set_limit(gas.limit().saturating_div(token_ratio_u64));
    }
    gas.set_refund(gas.refunded().saturating_mul(token_ratio_u64 as i64));
    gas.set_remaining(gas.remaining().saturating_mul(token_ratio_u64));
}
```

Tool Used

Manual Review

Recommendation

Validate and bound `token_ratio` once, then propagate a safe `ratio_u64` (default to 1 if zero/out-of-range). Replace all `unwrap()` and `assert!` with checked conversions and explicit errors, and guard all divisions. Consider keeping gas in gas units and applying `token_ratio` only to fee computations to avoid multi-point scaling.

If `token_ratio` is guaranteed non-zero by protocol, still keep these checks to contain faults from stale/corrupt reads and avoid panics in consensus-critical paths.

Discussion

RealiCZ

Keep the code consistent with op-geth.

OxEVom

It may be advisable to add a `require()` statement to `setTokenRatio()` that prevents this from happening, even unintentionally, seeing as both implementations would fail to handle it properly.

pandainzoo

this is a same question as

<https://github.com/sherlock-audit/2025-07-mantle/issues/302>

The update of the tokenRatio value is controlled by the off-chain service gasoracle. The calculation process is strictly constrained by minimum and maximum values, ensuring it does not reach zero in actual operation. However, adding a check at the contract level would be more prudent. We will consider incorporating this improvement in the next upgrade.

<https://github.com/mantlenetworkio/mantle-v2/blob/main/gas-oracle/tokenratio/tokenratio.go>
[allowbreak #L38](#)

Issue L-20: Wei vs gas units compared when effective gas price is zero can panic [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/448>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

`execution()/inspect_execution()` subtract an L1 fee (wei) from `gas_limit` (gas units) if `effective_gas_price == 0`, which can cause a panic or mis-accounting.

Vulnerability Detail

In non-deposit transactions, `crates/op-revm/src/handler.rs` `execution()` and `inspect_execution()` obtain the L1 data fee in wei from `calculate_tx_l1_cost()`, then only convert wei→gas by dividing with `effective_gas_price` when `effective_gas_price > 0`. If `effective_gas_price == 0`, the conversion is skipped and the wei-denominated `tx_l1_cost` is directly compared to and subtracted from `gas_limit` (u64 gas units) via `try_into().unwrap()`. This mixes dimensions and may panic or incorrectly gate execution. While mainnet validation enforces `effective_gas_price >= basefee`, this code path is still reachable in configurations where base-fee checks are disabled or basefee is zero.

Impact

If `effective_gas_price == 0` is permitted by configuration, comparing/subtracting wei from gas-limit units can trigger a panic or produce incorrect gas-limit enforcement, leading to node instability or inconsistent fee accounting in edge deployments.

Code Snippet

```
File: handler.rs
583:         let mut tx_l1_cost =
    ↪   ctx.chain().calculate_tx_l1_cost(&enveloped_tx, spec);
584:         let effective_gas_price = ctx.tx().effective_gas_price(basefee);
585:
586:         if effective_gas_price > 0 {
587:             tx_l1_cost =
    ↪   tx_l1_cost.wrapping_div(U256::from(effective_gas_price));
588:         }
589:
590:         if tx_l1_cost.gt(&U256::from(gas_limit)) {
591:             return Err(ERROR::from(OpTransactionError::Base(
592:                 InvalidTransaction::CallGasCostMoreThanGasLimit {
```

```

593:             initial_gas: init_and_floor_gas.initial_gas,
594:             gas_limit,
595:         },
596:     ));
597: }
598:
599:     gas_limit = gas_limit.wrapping_sub(tx_l1_cost.try_into().unwrap());

```

Tool Used

Manual Review

Recommendation

Always convert the L1 fee (wei) to gas units before comparisons, and reject non-deposit transactions with `effective_gas_price == 0` in these paths. Apply the same fix to inspection.

```

diff --git a/crates/op-revm/src/handler.rs b/crates/op-revm/src/handler.rs
@@ -1,7 +1,16 @@
-         let mut tx_l1_cost = ctx.chain().calculate_tx_l1_cost(&enveloped_tx,
+         spec);
+         let tx_l1_cost_wei = ctx.chain().calculate_tx_l1_cost(&enveloped_tx,
+         spec);
+         let effective_gas_price = ctx.tx().effective_gas_price(basefee);
-         if effective_gas_price > 0 {
-             tx_l1_cost =
+         tx_l1_cost.wrapping_div(U256::from(effective_gas_price));
-         }
+         if effective_gas_price == 0 {
+             return Err(ERROR::from(OpTransactionError::Base(InvalidTransaction
+         ::GasPriceLessThanBasefee)));
+         }
+         let tx_l1_cost_gas = tx_l1_cost_wei
+             .checked_div(U256::from(effective_gas_price))
+             .ok_or_else(||
+         ERROR::from(OpTransactionError::Base(InvalidTransaction::Overflow)))?;
+         let tx_l1_cost_gas_u64: u64 = tx_l1_cost_gas
+             .try_into()
+             .map_err(|_| ERROR::from(OpTransactionError::Base(
+         InvalidTransaction::CallGasCostMoreThanGasLimit { initial_gas:
+         init_and_floor_gas.initial_gas, gas_limit }
+             )))?;
-         if tx_l1_cost.gt(&U256::from(gas_limit)) {
+         if tx_l1_cost_gas_u64 > gas_limit {
+             return Err(ERROR::from(OpTransactionError::Base(

```

```
InvalidTransaction::CallGasCostMoreThanGasLimit { initial_gas:
    ↪ init_and_floor_gas.initial_gas, gas_limit },
    ));
}
- gas_limit = gas_limit.wrapping_sub(tx_l1_cost.try_into().unwrap());
+ gas_limit = gas_limit.saturating_sub(tx_l1_cost_gas_u64);
```

Discussion

0xEVom

@Realicz <https://github.com/mantle-xyz/revm/pull/10> does not seem to address this finding.

Realicz

@0xEVom Our current base fee is fixed at 0.02, so it will never be zero. This part of the code will be updated when we introduce EIP-1559 in the future to prevent any unexpected cases.

lpetroulakis

Acknowledged by the team without immediate remediation needed.

Issue L-21: Outdated repository/homepage metadata across repos [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/449>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The `repository`, `homepage` and similar fields in top-level `Cargo.toml` files across `op-succinct`, `kona`, and `revm` still point to upstream/original projects rather than the active fork, leading to misleading metadata.

Vulnerability Detail

- `op-succinct`: [Cargo.toml](#)
- `kona`: [Cargo.toml](#)
- `revm`: [Cargo.toml](#)

Impact

Tooling that uses manifest metadata (package indexes, SBOM generators, CI release builders, source-linking in error reports) will resolve to the wrong repositories or documentation.

Code Snippet

```
# op-succinct/Cargo.toml
homepage = "https://succinctlabs.github.io/op-succinct/"
repository = "https://github.com/succinctlabs/op-succinct"

# revm/Cargo.toml
repository = "https://github.com/bluealloy/revm"
documentation = "https://bluealloy.github.io/revm/"

# kona/Cargo.toml
homepage = "https://github.com/op-rs/kona"
repository = "https://github.com/op-rs/kona"
```

Tool Used

Manual Review

Recommendation

Update the metadata in each repo's top-level `Cargo.toml` to point at the forked organization and canonical docs site.

Discussion

RealiCZ

Acknowledged, it will be fixed in the next version

Issue L-22: Manifest pins diverge from audited commits across op-succinct, hydro and kona [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/451>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Three of the four repos are not importing each other at the audited commits; or dependencies are pinned to forks or branches, not the specific SHAs under review.

Vulnerability Detail

op-succinct, kona and hydro import revm from the skadi_new branch rather than the audited mantle-xyz/revm@59cbd48, so execution code is not the commit under review. op-succinct also imports hydro-* from PinelliaC/hydro@main instead of mantle-xyz/hydro@30346cb. kona consumers use the correct branch kona-client/v1.0.1_mantle, but do not pin to the audited commit 857afbf7, which allows branch drift away from the reviewed SHA. As a result, builds use a forked origin and an unpinned execution branch.

Impact

End-to-end builds may pull different code than the audited SHAs, invalidating review guarantees; API/behavior drift across forks or moving branches can introduce integration failures.

Code Snippet

```
# op-succinct/Cargo.toml - different hydro fork and revm branch
hydro-client = { git = "https://github.com/PinelliaC/hydro", branch = "main" }
hydro-host = { git = "https://github.com/PinelliaC/hydro", branch = "main" }
hydro-oracle = { git = "https://github.com/PinelliaC/hydro", branch = "main" }
hydro-eigenda = { git = "https://github.com/PinelliaC/hydro", branch = "main" }
hydro-proofs = { git = "https://github.com/PinelliaC/hydro", branch = "main" }

revm = { git = "https://github.com/mantle-xyz/revm", branch = "skadi_new",
  ↪ default-features = false, features = [
    "kzg-rs",
    "bn",
```

```
] }  
op-revm = { git = "https://github.com/mantle-xyz/revm", branch = "skadi_new",  
  ↪ default-features = false }
```

```
# hydro and kona - incorrect branch  
revm = { git = "https://github.com/mantle-xyz/revm", branch = "skadi_new",  
  ↪ default-features = false }  
op-revm = { git = "https://github.com/mantle-xyz/revm", branch = "skadi_new",  
  ↪ default-features = false }
```

Tool Used

Manual Review

Recommendation

Pin all cross-repo dependencies to the exact audited SHAs:

- Point hydro-* to mantle-xyz/hydro at 30346cb and remove forked origins.
- Pin all kona-* crates to mantle-xyz/kona@857afbf7.
- Pin revm/op-revm to mantle-xyz/revm@59cbd48.

Discussion

RealicZ

Although the branches differ, the reviewed content is consistent with what was expected. Once the new releases are available, we will switch to the corresponding tags.

Issue L-23: Deprecated EigenDA v1 integration may break data availability [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/452>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

EigenDA V1 is deprecated and being sunset; this codebase integrates V1 interfaces, risking sudden DA failures and incompatibilities without a migration to EigenDA Blazar (V2).

Vulnerability Detail

The DA flow integrates EigenDA via `EigenDAClient` and its methods `RetrieveBlob()` and `RetrieveBlobWithCommitment()`, which import `github.com/Layr-Labs/eigenda/api/grpc/disperser` (the V1 API). Derivation pulls EigenDA data in `dataFromEigenDa()`, while the Rust witness pipeline constructs an EigenDA source via `EigenDASource::new()`. The upstream spec [notes](#) that V1 is “deprecated and in the process of being completely sunset” and advises migration to EigenDA Blazar (V2). Continuing to rely on V1 endpoints and formats leaves DA retrieval and decoding subject to disruption when V1 services are removed or altered.

Impact

If V1 endpoints or formats are retired or diverge, EigenDA retrieval calls fail, `dataFromEigenDa()` cannot provide frames, and derivation/proving halts, preventing new L2 blocks from being produced or proven.

Code Snippet

```
import "github.com/Layr-Labs/eigenda/api/grpc/disperser"

func (c *EigenDAClient) RetrieveBlob(ctx context.Context, BatchHeaderHash []byte,
    ↪ BlobIndex uint32) ([]byte, error) {
    // ...
}
```

Tool Used

Manual Review

Recommendation

Migrate to EigenDA Blazar (V2) across the DA client and derivation pipeline: replace the V1 disperser client in `contracts/lib/mantle-v2/op-service/eigenda/da_proxy.go`, update frame retrieval/decoding in `dataFromEigenDa()`, and adapt witness generation/KZG verification to V2 semantics.

Discussion

RealicZ

The EigenDA team only recently released V2, and we are currently discussing the details of the upgrade to address this.

Oxbok

putting a note here so we don't forget. Issue needs to be merged with #286 or vice versa.

lpetroulakis

Acknowledged by the team.

Issue L-24: Trusted EigenDA integration reduces assurance by design (sequencer/proxy fully trusted) [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/455>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The rollup uses EigenDA's trusted integration model, which delegates certificate and recency checks to the sequencer/proxy and does not enforce independent verification in the derivation path. This is acceptable by design, but it lowers assurance and should be explicitly documented.

Vulnerability Detail

EigenDA's trusted integration “focuses on dispersal and retrieval for the sake of simplicity, but at the cost of security,” trusting the sequencer to verify certificates and post them to the inbox in a timely fashion. In this setup, the read path accepts EigenDA references without validating certificate authenticity or punctuality on its own. With proxy-side verification disabled, the system additionally trusts the disperser or transport to provide correct metadata. These choices match the trusted model but remove defense-in-depth: nodes do not anchor acceptance to the EigenDA Manager's onchain state and do not reject stale certificates.

Impact

If the sequencer or proxy misbehaves, nodes may accept uncertified or stale DA references and derive state that is not backed by EigenDA's certificate/recency guarantees. In the extreme, with fraud proofs disabled, a malicious sequencer can post arbitrary state roots without a practical path to challenge.

Tool Used

Manual Review

Recommendation

- Enable proxy certificate verification in production (set `EIGENDA_PROXY_EIGENDA_CERT_VERIFICATION_DISABLED=false`) to avoid trusting the disperser.

- Document the trusted model and surface it via logs/metrics.

Discussion

RealICZ

The eigenda-proxy is integrated as part of the Mantle operator node. The operator node has expanded from its original two components to three components: op-node, op-geth and eigenda-proxy.

Issue L-25: kona/crate/proof/mpt's unblind() doesn't check if the unblinded data hashes to the provided commitment [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/457>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

No verification that the fetched node actually hashes to the commitment. A malicious provider can return any node.

Vulnerability Detail

unblind() only relies on fetcher to retrieve the node preimage for commitment but doesn't verify the retrieved node hashes to commitment.

```
pub fn unblind<F: TrieProvider>(&mut self, fetcher: &F) -> TrieNodeResult<> {
    if let Self::Blinded { commitment } = self {
        if *commitment == EMPTY_ROOT_HASH {
            *self = Self::Empty;
        } else {
            *self = fetcher
                .trie_node_by_hash(*commitment)
                .map_err(|e| TrieNodeError::Provider(e.to_string()))?;
        }
    }
    Ok(())
}
```

Thus, it relies on this fetching mechanism to provide the correct preimage node.

Impact

Compromise of trie integrity. Malicious prover can forge invalid state.

Code Snippet

[proof/mpt/src/node.rs#L122-L135](#)

Tool Used

Manual Review

Recommendation

Verify the retrieved node corresponds to `commitment`.

Discussion

RealiCZ

`TrieProvider` is only used to fetch data from the locally stored `oracle`. This data has already been verified before being stored: https://github.com/mantle-xyz/kona/blob/kona-client/v1.0.1_mantle/crates/proof/proof/src/l1/chain_provider.rs [allowbreak #L126-L139 https://github.com/mantle-xyz/kona/blob/kona-client/v1.0.1_mantle/crates/proof/proof/src/l2/chain_provider.rs](https://github.com/mantle-xyz/kona/blob/kona-client/v1.0.1_mantle/crates/proof/proof/src/l2/chain_provider.rs) [allowbreak #L174-L189](https://github.com/mantle-xyz/kona/blob/kona-client/v1.0.1_mantle/crates/proof/proof/src/l2/chain_provider.rs)

Oxbok

I have set the severity as low/info. i'll suggest having this check as a defense mechanism, just like eigenDA suggests checking the blobs against commitment.

Issue L-26: TODO and legacy remarks left in production code paths [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/462>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Numerous TODO/FIXME and legacy remarks remain across the scoped repositories. While low severity, they indicate unfinished work or heuristics in critical paths. Below is a concise, line-linked index to prioritize cleanup.

Vulnerability Detail

- op-succinct
 - [utils/eigenda/host/src/host.rs#L66, #L78, #L88](#)
 - [utils/client/src/precompiles/mod.rs#L40](#)
- hydro
 - [bin/host/src/eigenda/handler.rs#L82, #L114, #L121, #L138](#)
 - [crates/oracle/src/provider.rs#L70](#)
 - [crates/proofs/src/witness.rs#L35, #L52](#)
 - [crates/eigenda/src/derive/eigenda.rs#L277](#)
 - [crates/eigenda/src/common/constant.rs#L4](#)
 - [crates/eigenda/src/common/certificate.rs#L5](#)
- revm
 - [crates/op-revm/src/transaction/abstraction.rs#L153](#)
 - [crates/op-revm/src/handler.rs#L124](#)
 - Additional EOF-related TODOs in interpreter/handler modules are present throughout the revm workspace.
- kona
 - [crates/protocol/derive/src/stages/batch/batch_provider.rs#L134](#)
 - [crates/supervisor/service/src/service.rs#L14, #L27, #L84](#)
 - [crates/supervisor/core/src/state/mod.rs#L2](#)
 - Additional TODOs in node RPC/types and proof boot config files remain.

Impact

Leaving unresolved markers in production code decreases maintainability and increases the risk of latent bugs.

Code Snippet

```
// utils/eigenda/host/src/host.rs
/// [TODO] CHECK
async fn get_finalized_l2_block_number(...) -> Result<Option<u64>> { ... }
```

Tool Used

Manual Review

Recommendation

Resolve or remove TODO/FIXME in production paths, or gate behind feature flags with linked issue references. Introduce CI to prevent unscoped TODOs in non-test code.

Discussion

RealiCZ

Acknowledged, it will be fixed in the next version

Issue L-27: Precompile input extraction silently converts errors to empty input [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/463>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

When extracting input data from shared memory buffers in precompile calls, the code silently defaults to empty bytes on failure, potentially masking critical errors and causing incorrect execution.

Vulnerability Detail

In the `dispatch_call()` function, when processing precompile inputs that use a `SharedBuffer` variant, the code calls `context.local().shared_memory_buffer_slice()` which can fail and return `None` if:

- The provided range is out of bounds for the shared memory buffer
- The buffer cannot be borrowed (e.g., if it's already mutably borrowed elsewhere)

When this operation fails, instead of propagating an error, the code uses `unwrap_or_default()` which silently converts the failure into an empty `Bytes::new()`. This means the precompile would execute with empty input data rather than the intended input, without any indication that an error occurred.

While such failures should be extremely rare in normal operation (as the EVM validates and pre-expands memory before creating these ranges), if they do occur due to bugs in memory management or state corruption, the silent conversion makes debugging difficult and could lead to incorrect state derivation compared to L1.

Impact

If a memory extraction failure occurs, the precompile would execute with empty input instead of the actual call data, causing the zkVM's state computation to diverge from L1. This would produce an invalid proof that would be rejected during on-chain verification.

Code Snippet

```
use revm::context::LocalContextTr;  
let input = match &inputs.input {  
    revm::interpreter::CallInput::Bytes(bytes) => bytes.clone(),  
    revm::interpreter::CallInput::SharedBuffer(range) => context
```

```
.local()  
.shared_memory_buffer_slice(range.clone())  
.map(|b| Bytes::from(b.to_vec()))  
.unwrap_or_default(), // Silently converts error to empty input  
};
```

Tool Used

Manual Review

Recommendation

Propagate the error explicitly when `shared_memory_buffer_slice()` fails rather than defaulting to empty bytes.

Discussion

Realicz

Forked from Succinct code, to be fixed in the next version.

Issue L-28: Silent truncation accepted during blob decode [RESOLVED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/465>

Summary

`decode()` permits length mismatches by accepting a smaller declared `content_size` than the actual decoded payload, returning a truncated slice instead of rejecting malformed input.

Vulnerability Detail

The `EigenDABlobData::decode()` implementation in `crates/eigenda/src/common/eigenda_data.rs` reads the declared length `content_size` from the header and derives the decoded payload `blob_content`. It only errors when `blob_content.len() < content_size`, then unconditionally returns `blob_content[..content_size]`. This accepts inputs where `content_size` is smaller than the actual decoded payload, silently truncating data rather than rejecting a malformed blob. The referenced Go implementation uses stricter validation that would not accept this mismatch.

Impact

A malformed blob with an understated `content_size` is treated as valid and decoded to truncated data, leading to silent data loss and cross-implementation inconsistency.

Code Snippet

```
let content_size = blob.slice(2..6).get_u32();
let codec_data = blob.slice(32..);
let blob_content =
    helpers::remove_empty_byte_from_padded_bytes_unchecked(codec_data.as_ref());
let blob_content: Bytes = blob_content.into();

if blob_content.len() < content_size as usize {
    return Err(BlobDecodingError::InvalidLength);
}
Ok(blob_content.slice(..content_size as usize))
```

Tool Used

Manual Review

Recommendation

Enforce strict length validation by rejecting any mismatch between `content_size` and the decoded payload length. This aligns behaviour with the Go implementation and prevents silent truncation.

Discussion

byteflyfunny

will be fixed pr

0xEVom

@byteflyfunny @RealICZ noting here that <https://github.com/mantle-xyz/hydro/pull/3> adds additional validations (length multiple-of-32, field element/encoding constraints), but does not change the leniency that allows truncation when decoded length exceeds `content_size`.

Issue L-29: No limit on EigenDA data_length [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/466>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

data_length from the operator-authored EigenDA certificate is used to size allocations and loop bounds without limits in the oracle provider.

Vulnerability Detail

In `OracleEigenDaProvider::blob_get()`, the function parses the EigenDA certificate carried in the L1 batcher transaction calldata and uses `cert_blob_info.blob_header.data_length` to size a buffer and to drive a per-element loop. The first use occurs in [crates/oracle/src/provider.rs#L63](#), allocating `data_length * 32` bytes. The loop bound is taken at [crates/oracle/src/provider.rs#L86](#). There is no explicit upper bound or overflow guard before allocation and iteration.

Impact

A large data_length can trigger oversized allocations or long iteration. In the trusted model this is low likelihood, but the consequence is a local denial of service for the node.

Code Snippet

```
// crates/oracle/src/provider.rs
let mut blob: Vec<u8> =
    vec![0; cert_blob_info.blob_header.data_length as usize *
        BYTES_PER_FIELD_ELEMENT];

let data_length = cert_blob_info.blob_header.data_length as u64;

for i in 0..data_length {
    // hashing + oracle calls per iteration
}
```

Tool Used

Manual Review

Recommendation

Set a fixed max for `data_length` so both allocation and the loop are bounded; use checked multiplication for the buffer size.

Discussion

byteflyfunny

The size of each submission to EigenDA is not fixed. If the zkVM on the Succinct side has memory limitations, we need to consider setting an upper bound during OP-batcher rollup. For now, this issue will not be fixed, but it will be continuously tracked.

Issue L-30: Redundant SRS loading on every witness generation [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/468>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The SRS is loaded from disk on every witness generation call, loading 1,000,000 points despite only needing up to 131,072.

Vulnerability Detail

The `push_witness()` function loads the entire SRS from disk every time it's called:

1. It loads 1,000,000 G1 points from `resources/g1.point` even though the KZG domain is capped at `MAINNET_SRS_G1_SIZE` (131,072 points). Approximately 87% of loaded data is never used.
2. The SRS is immutable and could be cached at startup, but instead triggers ~32MB of disk I/O on every call.

Impact

Each witness computation unnecessarily loads and deserializes 1,000,000 elliptic curve points from disk, causing avoidable I/O overhead and CPU processing.

Code Snippet

```
// crates/proofs/src/witness.rs:34-42
pub fn push_witness(&mut self, blob: &[u8]) -> Result<(), KzgError> {
    // TODO remove the need for G2 access
    // Add command line to specify where are g1 and g2 path
    // In the future, it might make sense to let the proxy to return such
    // value, instead of local computation
    let srs = SRS::new("resources/g1.point", 268435456, 1000000).unwrap();
    let mut kzg = KZG::new();
    // ...
}
```

Tool Used

Manual Review

Recommendation

Cache the SRS at initialization and load only the required number of points:

```
+ use std::sync::Arc;
+ use once_cell::sync::OnceCell;
+
+ static SRS_CACHE: OnceCell<Arc<SRS>> = OnceCell::new();
+
+ fn get_cached_srs() -> Result<Arc<SRS>, KzgError> {
+     SRS_CACHE.get_or_try_init(|| {
+         // Load only the points we actually need (MAINNET_SRS_G1_SIZE)
+         let srs = SRS::new("resources/g1.point", 268435456, 131072)?;
+         Ok(Arc::new(srs))
+     }).cloned()
+ }

pub fn push_witness(&mut self, blob: &[u8]) -> Result<(), KzgError> {
-     let srs = SRS::new("resources/g1.point", 268435456, 1000000).unwrap();
+     let srs = get_cached_srs()?;
    let mut kzg = KZG::new();
```

Discussion

RealiCZ

Acknowledged, we will adopt LazyLock to reduce the loading times.

```
+use std::sync::LazyLock;
+
+static GLOBAL_SRS: LazyLock<SRS> = LazyLock::new(|| {
+     SRS::new("resources/g1.point", 268435456, 1000000)
+         .expect("Failed to initialize SRS")
+ });
```

Issue L-31: EIP-4844 blobs are not bound to requested hashes in online mode [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/470>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

In online mode, fetched EIP-4844 blobs are turned into commitments/proofs and stored without checking they match the originally requested `IndexedBlobHash.hash`, weakening integrity binding to the request.

Vulnerability Detail

In `OnlineBlobStore::get_blobs()`, the code fetches blobs from the underlying provider, then iterates the returned list and computes a KZG commitment/proof for each via `get_blob_data()`. The results are appended into `BlobData (store.blobs, store.commitments, store.proofs)` with no verification that the *i*-th blob corresponds to the *i*-th `IndexedBlobHash.hash` requested, nor any mapping by hash. If the provider reorders or returns incorrect blobs, the witness ends up containing commitments/proofs for whatever bytes were returned, not necessarily the requested versioned hashes.

Impact

A misordered or incorrect set of returned blobs can be accepted and committed locally, allowing later verification to succeed against self-derived commitments while not actually matching the requested `IndexedBlobHash.hash` set.

Code Snippet

```
File: online_blob_store.rs
32:         for blob in &blobs {
33:             let (c_kzg_blob, commitment, proof) = get_blob_data(blob,
    ↪ &settings);
34:
35:             store.blobs.push(c_kzg_blob);
36:             store.commitments.push(commitment);
37:             store.proofs.push(proof);
38:         }
```

Tool Used

Manual Review

Recommendation

Bind each returned blob to its requested `IndexedBlobHash.hash`: compute the versioned hash from the blob's KZG commitment and assert it matches the requested hash, or build a hash→blob map and populate `BlobData` in that order.

```
        let mut store = self.store.lock().unwrap();
-        for blob in &blobs {
+        for (i, blob) in blobs.iter().enumerate() {
            let (c_kzg_blob, commitment, proof) = get_blob_data(blob, &settings);
+            // derive versioned hash from commitment and ensure it matches the
↪ request
+            let got =
↪ alloy_eips::eip4844::kzg_to_versioned_hash(commitment.as_ref());
+            let want = blob_hashes[i].hash;
+            if got != want {
+                return Err( anyhow::anyhow!("blob/versioned-hash mismatch at index
↪ {i}"));
+            }
            store.blobs.push(c_kzg_blob);
            store.commitments.push(commitment);
            store.proofs.push(proof);
        }
```

Discussion

byteflyfunny

The Succinct code can be considered for a fix in a future version.

Issue L-32: Panics on malformed DA inputs and RPC responses can terminate the host/derivation [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/471>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Several DA ingestion paths use `unwrap()`/`expect()` on untrusted inputs. Malformed EIP-4844 blobs or missing/invalid RPC responses can panic the process and interrupt proving or derivation.

Vulnerability Detail

Data availability inputs include EIP-4844 blobs and network RPC responses used by the host and derivation flows. In multiple places, conversions and lookups are performed using `unwrap()`/`expect()`. If blob bytes are malformed or RPC calls return `None`/errors, these calls will panic rather than return an error, terminating the process.

Key instances in DA-related code paths:

- op-succinct: 4844 online path (KZG conversions/proofs)
 - `get_blob_data()` in `utils/host/src/witness_generation/online_blob_store.rs`:
 - * `c_kzg::Blob::from_bytes(...).unwrap()`; commitment/proof generation `.unwrap()`s; final `KzgRsBlob::from_slice(...).unwrap()`
- op-succinct: 4844 client-side blob handling
 - `impl From<BlobData> for BlobStore` in `utils/client/src/oracle/blob_provider.rs`:
 - * `kzg_rs::Blob::from_slice(...).unwrap()` on incoming blob bytes
 - * `KzgProof::verify_blob_kzg_proof_batch(...).expect(...)` panics on verification failure
 - * `self.versioned_blobs.pop().unwrap()` when serving blobs
- op-succinct: Host RPC-based DA/witness fetching
 - `utils/host/src/fetcher.rs`:
 - * `get_block_receipts(...).await.unwrap()` / `receipt.unwrap()`
 - * `get_block_by_number(...).await?.unwrap()` and `receipts.unwrap()`

- * earliest_ll_header.unwrap()
 - * get_block_by_number(...).await?.unwrap()
- op-succinct: Witness pipeline input collection
 - `utils/host/src/witness_generation/traits.rs`:
 - * get_inputs_for_pipeline(...).await.unwrap()
- hydro: EigenDA certificate decoding and witness utilities
 - `crates/oracle/src/provider.rs`:
 - * BlobInfo::decode(mut commitment[3..]).unwrap() on proxy/cert bytes
 - `crates/proofs/src/witness.rs`:
 - * SRS::new(...).unwrap(); calculate_and_store_roots_of_unity(...).unwrap() on local SRS/BN254 setup
 - `bin/host/src/eigenda/handler.rs`:
 - * assert!(eigenda_blob.blob.len() <= blob_length as usize * BYTES_PER_FIELD_ELEMENT, ...) on untrusted size

Impact

A malformed 4844 blob or missing/invalid RPC response can panic and terminate host/derivation, interrupting proving or data generation. This is a low-severity availability risk.

Recommendation

Replace `unwrap()/expect()` in DA ingestion with explicit error handling and precondition checks; for RPC calls, use timeouts/retries and propagate errors. Validate blob sizes before KZG conversions and avoid panicking when serving blobs or decoding EigenDA certificates.

Discussion

byteflyfunny

To be fixed in the next version

Issue L-33: Range splitting can panic on a single RPC failure [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/472>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

`split_range_based_on_safe_heads()` uses a hard `.expect(...)` inside a concurrent RPC loop and scans an unbounded L1 interval, allowing any transient RPC error to crash the span-batching step.

Vulnerability Detail

The helper `split_range_based_on_safe_heads()` iterates the L1 block range from the L1 origin of the starting L2 block to the L1 block from which the end L2 block can be derived. For each L1 block height, it requests `optimism_safeHeadAtL1Block` and collects the resulting L2 safe heads to determine span boundaries. Requests are batched with `.buffered(15)`, but inside the async mapper the code calls `.expect("Failed to fetch safe head")`. Any single failure (timeout, provider hiccup) panics and aborts the computation.

Impact

A transient RPC failure during span boundary discovery aborts the proof range planning with a panic, interrupting data generation. When the proving window covers many L1 blocks, the probability of a failure increases.

Code Snippet

```
let safe_heads = futures::stream::iter(l1_start..=l1_head_number)
    .map(|block| async move {
        let l1_block_hex = format!("0x{block:x}");
        let data_fetcher = OPSuccinctDataFetcher::default();
        let result: SafeHeadResponse = data_fetcher
            .fetch_rpc_data_with_mode(
                RPCMode::L2Node,
                "optimism_safeHeadAtL1Block",
                vec![l1_block_hex.into()],
            )
            .await
            .expect("Failed to fetch safe head"); // panics on error
        result.safe_head.number
    })
```

```
.buffered(15)
.collect::
```

Tool Used

Manual Review

Recommendation

Replace `.expect(...)` with structured error handling and retries/backoff. Consider paginating the LI scan into fixed-size pages so a single page failure does not crash the entire operation, and allow resuming from the last successful page.

Discussion

byteflyfunny

Forked from Succinct code, to be fixed in the next version.

Issue L-34: Bedrock simplifications and half-removed OP Stack guards should be config-gated and cleaned up [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/473>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Multiple Bedrock simplifications and OP Stack guard removals exist across Kona and op_{succinct}, with some related code paths left half_{removed}. While safe for the current deployment, these deviations should be config_{gated} and cleaned up to avoid future misconfiguration and drift.

Vulnerability Detail

Below is a non-exhaustive list of instances that change canonical behavior or remove guards:

- Fork-gated header fields forced “always-set”
 - Withdrawals root always = L2→L1 Message Passer storage root: [assemble.rs#L54-L63](#), [#L96-L105](#)
 - Requests hash always = empty SHA₂₅₆: [assemble.rs#L90-L95](#)
- Span batching/mux relaxations
 - `is_last_in_span()` hardcoded true: [batch_provider.rs#L133-L136](#)
 - Mux simplified to `BatchQueue`: [batch_provider.rs#L63-L66](#)
- Fork parameter/upgrade handling disabled
 - Holocene encode/decode + Cancun blob gas logic removed: [assemble.rs \(deletions\)](#), [env.rs \(deletions\)](#)
- DA selection/fallback removal
 - Celestia path removed (EigenDA only): [op_{succinct} compare](#)
- Interop activation/registry guards removed (interop off)
 - Only expiry/origin/hash/timestamp checks remain: [graph.rs#L146-L157](#), [#L159-L211](#)
- Protocol version runtime checks removed
 - L1 protocolversions slot reads deleted: [runtime/loader.rs \(deletions\)](#)
- Dead and half-removed code (risk of accidental re_{enablement}/drift)

- Holocene param helpers left in tree but unused:
 - * `decode_holocene_eip_1559_params()` and `encode_holocene_eip_1559_params()` are annotated with `#[allow(dead_code)]`: [util.rs#L19-L21](#), [#L56-L60](#)
 - * References commented out in builders/env: [env.rs \(comments\)](#), [assemble.rs \(comments\)](#)
- Protocol version machinery retained in RPC/types but enforcement removed in loader:
 - * Types/API still present: [node/rpc/src](#) (e.g., `ProtocolVersion`), while loader's onchain reads are commented: [runtime/loader.rs \(comments\)](#)
- Span queue state fields/methods removed in core but some test scaffolding and assumptions remain elsewhere (see engine tests and p2p payload v4 paths).

Impact

Low: internally consistent for a Bedrock-like network, but diverges from OP Stack canon and reduces guardrails. Risk is primarily future-compatibility and misconfiguration if features are re-enabled without restoring checks; dead/half-removed code increases the chance of accidental re-activation or silent drift.

Recommendation

Config-gate these behaviors and keep tests for both simplified (Mantle) and canonical paths. Remove or isolate dead helpers behind feature flags; add assertions/logs to detect unintended activation.

Discussion

byteflyfunny

will be fixed next version

Issue L-35: Unbounded HTTP body read can exhaust memory [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/477>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The EigenDA proxy client buffers entire HTTP responses without a size cap, allowing a large or streaming body to exhaust memory. This is low severity because the proxy is trusted, but it still widens the blast radius of misconfiguration or proxy compromise.

Vulnerability Detail

`EigenDAProxy::retrieve_blob_with_commitment` fetches blobs from the proxy and, on `200 OK`, calls `response.bytes().await` to materialise the entire body into memory as a `Vec<u8>`. There is no check on `Content-Length`, no streaming with a hard cap, and no enforcement of an upper bound derived from the commitment's expected maximum blob size. The exact unbounded read occurs at [online_provider.rs#L49-L53](#).

A malicious or misconfigured proxy (or an unintended large response) can return arbitrarily large bodies or an infinite stream; the client will attempt to buffer it fully, risking out-of-memory conditions.

Impact

A large or unbounded HTTP response causes the host process to allocate until memory exhaustion, resulting in process termination and denial of service for proof generation or data ingestion.

Code Snippet

```
match response.status() {
    StatusCode::OK => response
        .bytes()
        .await
        .map(|bytes| bytes.to_vec())
        .map_err(|e| EigenDAProxyError::RetrieveBlobWithCommitment(e.to_string())),
    ...
}
```

Tool Used

Manual Review

Recommendation

Stream the body with an explicit maximum and reject responses exceeding the cap; optionally, derive the cap from the commitment's declared `data_length * BYTES_PER_FIELDED_ELEMENT`, or use a conservative configured limit.

Discussion

byteflyfunny

will be optimized in next version

Issue L-36: No FrameRef context validation on EigenDA references [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/478>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

FrameRef context fields are not validated during EigenDA frame resolution. The implementation only checks `quorum_ids` and fetches by `commitment`, ignoring `batch_header_hash`, `blob_index`, and `reference_block_number`, weakening binding to the intended batch/L1 context.

Vulnerability Detail

In Hydro's EigenDA path, `EigenDASource::data_from_eigen_da()` handles `calldata_frame: Value::FrameRef` by warning (fix: reverting) if `quorum_ids` is empty and then immediately fetching by `commitment`, without validating the rest of the context.

The protobuf-compiled `FrameRef` includes the ignored context fields: `batch_header_hash`, `blob_index`, `reference_block_number`, as well as `blob_length` and `commitment`.

Because these fields are not checked, derivation accepts bytes solely by resolving a `commitment`, without asserting the batch header, position, or anchoring L1 block number. This omits a useful invariant for OP Stack-style batching: that the reference context matches the batch/L1 context the batcher intended.

Impact

Frames may be accepted under the wrong batch or from a stale L1 epoch, causing the prover to derive an L2 payload that does not correspond to the intended batch context. This risks generating proofs over inconsistent inputs and can stall proving if data is outside EigenDA's retention window.

Code Snippet

```
// crates/eigenda/src/derive/eigenda.rs#L146-L161
calldata_frame::Value::FrameRef(frame_ref) => {
    if frame_ref.quorum_ids.is_empty() {
        warn!(target: "eigen-da-source", "decoded frame ref contains no quorum
        ↪ IDs");
        continue;
    }
    // No validation of batch_header_hash, blob_index, reference_block_number
```

```

    let blob_data = self
        .eigen_da_provider
        .blob_get(&frame_ref.commitment)
        .await
        .map_err(|e| EigenDAProviderError::Status(e.to_string()))?;
    let blobs = &blob_data[..frame_ref.blob_length as usize];
    let blob_data: VecOfBytes = decode(blobs)?;
    for blob in blob_data.0 {
        data.push(Bytes::from(blob));
    }
}

```

Tool Used

Manual Review

Recommendation

- Decode BlobInfo from frame_ref.commitment before fetching bytes.
- Validate FrameRef context against BlobInfo (batch_header_hash, blob_index, reference_block_number).
- Enforce a punctuality window in load_blobs() using block_ref.number minus reference_block_number (e.g., STALE_GAP).
- Reject on any mismatch or stale reference; only then call blob_get() and decode frames.

Discussion

byteflyfunny

I understand that EigenDA's commitment generation is based on these parameters, so verifying the commitment is sufficient.

0xEVom

Yes, this is just a sanity check/defence in depth recommendation.

lpetroulakis

Acknowledged without any need to remediate.

Issue L-37: Changing IP address can circumvent peer banning [ACKNOWLEDGED]

Source: <https://github.com/sherlock-audit/2025-07-mantle/issues/453>

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

Discovery protocol may not be able to ban peers identified by Gossip protocol due to different Id systems used.

Vulnerability Detail

When an address is banned in `discv5/driver.rs`, it searches through ENRs using the multiaddress (which includes IP).

In a realistic scenario, a node will be able to update enr record to change its IP.

It is possible to update IP for an ENR record in discovery service, but the gossip service keeps using the same IP. Thus when gossip service tries to ban an IP and sends this request to discovery service (linked above), the discovery service may not be able to find that IP in its ENR records and thus unable to ban it.

Impact

Nodes can circumvent banning.

Code Snippet

`p2p/src/discv5/driver.rs#L341-L352:`

```
HandlerRequest::BanAddrs{addrs_to_ban, ban_duration} => {
    let enrs = self.disc.table_entries_enr();

    for enr in enrs {
        let Some(multi_addr) = enr_to_multiaddr(&enr) else {
            continue;
        };

        if addrs_to_ban.contains(&multi_addr) {
            self.disc.ban_node(&enr.node_id(), Some(ban_duration));
        }
    }
}
```

[p2p/src/net/driver.rs#L145](#)

```
if let Err(send_err) = handler.sender.send(HandlerRequest::BanAddrs {
    ↪   addr_to_ban: ban_peers.ban_duration }, ban_duration: ban_peers.ban_duration ).await{
    warn!(err = ?send_err, "Impossible to send a request to the discovery handler.
    ↪   The channel connection is dropped.");
}
```

Tool Used

Manual Review

Recommendation

Use `PeerId` to avoid including IP address in peer representation.

Discussion

RealiCZ

In Kona, we don't use the `node` crate, we plan to do a rebase once Kona makes their next release.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.