SHERLOCK

Security Review For Irys

Collaborative Audit Prepared For:	Irys
Lead Security Expert(s):	berndartmueller sammy sh1v
Date Audited:	June 2 - July 21, 2025
Final Commit:	c5aeab2

Introduction

Security review of the mainnet node. Its goal is to test that the node is performant and safe/ready for a mainnet launch.

Scope

Repository: Irys-xyz/irys

Audited Commit: db788b8c13071602a1f9db93b347ed51b712054d

Final Commit: c5aeab259213f1125cea1a6c0a6582f8450610e1

Files:

- crates/actors/Cargo.toml
- crates/actors/src/addresses.rs
- crates/actors/src/block_discovery.rs
- crates/actors/src/block_index_service.rs
- crates/actors/src/block_producer.rs
- crates/actors/src/block_tree_service.rs
- crates/actors/src/block_validation.rs
- crates/actors/src/broadcast_mining_service.rs
- crates/actors/src/cache_service.rs
- crates/actors/src/chunk_migration_service.rs
- crates/actors/src/commitment_cache.rs
- crates/actors/src/ema_service.rs
- crates/actors/src/epoch_service/commitment_state.rs
- crates/actors/src/epoch_service/epoch_replay_data.rs
- crates/actors/src/epoch_service/epoch_service_messages.rs
- crates/actors/src/epoch_service/epoch_service.rs
- crates/actors/src/epoch_service/mod.rs
- crates/actors/src/epoch_service/partition_assignments.rs
- crates/actors/src/lib.rs
- crates/actors/src/mempool_service.rs
- crates/actors/src/mining.rs
- crates/actors/src/packing.rs

- crates/actors/src/reth_service.rs
- crates/actors/src/services.rs
- crates/actors/src/storage_module_service.rs
- crates/actors/src/validation_service.rs
- crates/actors/src/vdf_service.rs
- crates/actors/tests/chunk_migration_test.rs
- crates/actors/tests/epoch_service_tests.rs
- crates/api-client/Cargo.toml
- crates/api-client/src/lib.rs
- crates/api-server/Cargo.toml
- crates/api-server/src/error.rs
- crates/api-server/src/lib.rs
- crates/api-server/src/routes/block_index.rs
- crates/api-server/src/routes/block.rs
- crates/api-server/src/routes/commitment.rs
- crates/api-server/src/routes/get_chunk.rs
- crates/api-server/src/routes/index.rs
- crates/api-server/src/routes/mod.rs
- crates/api-server/src/routes/network_config.rs
- crates/api-server/src/routes/peer_list.rs
- crates/api-server/src/routes/post_chunk.rs
- crates/api-server/src/routes/post_version.rs
- crates/api-server/src/routes/price.rs
- crates/api-server/src/routes/proxy.rs
- crates/api-server/src/routes/tx.rs
- crates/c/build/capacity.rs
- crates/c/build/main.rs
- crates/c/build/vdf.rs
- crates/c/Cargo.toml
- crates/c/c_src/capacity_cuda.cu
- crates/c/c_src/capacity_cuda.h

- crates/c/c_src/capacity.h
- crates/c/c_src/capacity_hip.hip
- crates/c/c_src/capacity_single.c
- crates/c/c_src/Makefile
- crates/c/c_src/seed_hash.c
- crates/c/c_src/sha256.cuh
- crates/c/c_src/tests/tests.cpp
- crates/c/c_src/types.h
- crates/c/.gitignore
- crates/chain/Cargo.toml
- crates/chain/src/chain.rs
- crates/chain/src/genesis_utilities.rs
- crates/chain/src/lib.rs
- crates/chain/src/main.rs
- crates/chain/src/peer_utilities.rs
- crates/chain/src/vdf.rs
- crates/chain/tests/api/api.rs
- crates/chain/tests/api/client.rs
- crates/chain/tests/api/external_api.rs
- crates/chain/tests/api/mod.rs
- crates/chain/tests/api/pricing_endpoint.rs
- crates/chain/tests/api/tx_commitments.rs
- crates/chain/tests/api/tx.rs
- crates/chain/tests/block_production/analytics.rs
- crates/chain/tests/block_production/basic_contract.rs
- crates/chain/tests/block_production/block_production.rs
- crates/chain/tests/block_production/mod.rs
- crates/chain/tests/block_production/testing_primitives.rs
- crates/chain/tests/ema_pricing/ema_pricing.rs
- crates/chain/tests/ema_pricing/mod.rs
- crates/chain/tests/external/api.rs

- crates/chain/tests/external/block_production.rs
- crates/chain/tests/external/mod.rs
- crates/chain/tests/external/programmable_data_basic.rs
- crates/chain/tests/external/readme.md
- crates/chain/tests/integration/cache_service.rs
- crates/chain/tests/integration/mod.rs
- crates/chain/tests/mod.rs
- crates/chain/tests/multi_node/fork_recovery.rs
- crates/chain/tests/multi_node/mempool_tests.rs
- crates/chain/tests/multi_node/mod.rs
- crates/chain/tests/multi_node/peer_discovery.rs
- crates/chain/tests/multi_node/peer_mining.rs
- crates/chain/tests/multi_node/sync_chain_state.rs
- crates/chain/tests/programmable_data/basic.rs
- crates/chain/tests/programmable_data/mod.rs
- crates/chain/tests/promotion/data_promotion_basic.rs
- crates/chain/tests/promotion/data_promotion_double.rs
- crates/chain/tests/promotion/mod.rs
- crates/chain/tests/startup/mod.rs
- crates/chain/tests/startup/startup.rs
- crates/chain/tests/synchronization/mod.rs
- crates/chain/tests/utils.rs
- crates/cli/Cargo.toml
- crates/cli/src/main.rs
- crates/config/Cargo.toml
- crates/config/src/chain/chain.rs
- crates/config/src/chain/chainspec.rs
- crates/config/src/chain/mod.rs
- crates/config/src/lib.rs
- crates/config/src/submodules.rs
- crates/c/src/capacity_cuda.rs

- crates/c/src/capacity.rs
- crates/c/src/capacity_single.rs
- crates/c/src/lib.rs
- crates/c/src/vdf.rs
- crates/database/Cargo.toml
- crates/database/examples/dupsort.rs
- crates/database/src/block_index_data.rs
- crates/database/src/config.rs
- crates/database/src/database.rs
- crates/database/src/data_ledger.rs
- crates/database/src/db_cache.rs
- crates/database/src/db_index.rs
- crates/database/src/db.rs
- crates/database/src/lib.rs
- crates/database/src/metadata.rs
- crates/database/src/migration.rs
- crates/database/src/reth_ext.rs
- crates/database/src/submodule/db.rs
- crates/database/src/submodule/mod.rs
- crates/database/src/submodule/tables.rs
- crates/database/src/system_ledger.rs
- crates/database/src/tables.rs
- crates/efficient-sampling/Cargo.toml
- crates/efficient-sampling/src/lib.rs
- crates/irys-reth/Cargo.toml
- crates/irys-reth/src/lib.rs
- crates/irys-reth/src/system_tx.rs
- crates/p2p/Cargo.toml
- crates/p2p/src/block_pool_service.rs
- crates/p2p/src/cache.rs
- crates/p2p/src/gossip_client.rs

- crates/p2p/src/gossip_service.rs
- crates/p2p/src/lib.rs
- crates/p2p/src/peer_list.rs
- crates/p2p/src/server_data_handler.rs
- crates/p2p/src/server.rs
- crates/p2p/src/sync.rs
- crates/p2p/src/tests/block_pool/mod.rs
- crates/p2p/src/tests/integration/mod.rs
- crates/p2p/src/tests/mod.rs
- crates/p2p/src/tests/util.rs
- crates/p2p/src/types.rs
- crates/p2p/src/vdf_utils.rs
- crates/packing/Cargo.toml
- crates/packing/src/lib.rs
- crates/price-oracle/Cargo.toml
- crates/price-oracle/src/lib.rs
- crates/primitives/Cargo.toml
- crates/primitives/src/commitment.rs
- crates/primitives/src/lib.rs
- crates/primitives/src/precompile.rs
- crates/primitives/src/range_specifier.rs
- crates/reth-node-bridge/Cargo.toml
- crates/reth-node-bridge/src/adapter.rs
- crates/reth-node-bridge/src/ext.rs
- crates/reth-node-bridge/src/lib.rs
- crates/reth-node-bridge/src/node.rs
- crates/reth-node-bridge/src/signal.rs
- crates/reth-node-bridge/src/unwind.rs
- crates/reward-curve/Cargo.toml
- crates/reward-curve/src/lib.rs
- crates/reward-curve/src/main.rs

- crates/storage/Cargo.toml
- crates/storage/src/chunk_provider.rs
- crates/storage/src/irys_consensus_data_db.rs
- crates/storage/src/lib.rs
- crates/storage/src/reth_provider.rs
- crates/storage/src/storage_module.rs
- crates/storage/tests/storage_module_index_tests.rs
- crates/types/Cargo.toml
- crates/types/configs/testnet.toml
- crates/types/src/app_state.rs
- crates/types/src/arbiter_handle.rs
- crates/types/src/block_production.rs
- crates/types/src/block.rs
- crates/types/src/chunked.rs
- crates/types/src/chunk.rs
- crates/types/src/config.rs
- crates/types/src/difficulty_adjustment_config.rs
- crates/types/src/gossip.rs
- crates/types/src/h256.rs
- crates/types/src/ingress.rs
- crates/types/src/irys.rs
- crates/types/src/lib.rs
- crates/types/src/merkle.rs
- crates/types/src/partition.rs
- crates/types/src/peer_list.rs
- crates/types/src/serialization.rs
- crates/types/src/signature.rs
- crates/types/src/simple_rng.rs
- crates/types/src/storage_pricing.rs
- crates/types/src/storage.rs
- crates/types/src/transaction.rs

- crates/types/src/version.rs
- crates/vdf/Cargo.toml
- crates/vdf/src/lib.rs
- fixtures/contracts/foundry.toml
- fixtures/contracts/.gitignore
- fixtures/contracts/out/IrysERC20.sol/IrysERC20.json
- fixtures/contracts/out/IrysProgrammableDataBasic.sol/ProgrammableDataBasic.json
- fixtures/contracts/out/ProgrammableDataLib.sol/ProgrammableData.json
- fixtures/contracts/out/ProgrammableData.sol/ProgrammableData.json
- fixtures/contracts/README.md
- fixtures/contracts/src/IrysERC20.sol
- fixtures/contracts/src/IrysProgrammableDataBasic.sol
- fixtures/contracts/src/Precompiles.sol
- fixtures/contracts/src/ProgrammableData.sol
- xtask/src/main.rs

Final Commit Hash

c5aeab259213f1125cea1a6c0a6582f8450610e1

Findings

Each issue has an assigned severity:

- 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.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- 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
16	8	11

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: Submitting a chunk with `data_size = 0` causes a panic during chunk processing

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/207

Summary

A submitted chunk with `data_size = 0` will cause a panic during chunk processing, leading to a denial of service.

Vulnerability Detail

`data_size = 0` is an illegal state, which is asserted in the data_size_to_chunk_count function. However, besides this assertion, no further validations are in place when processing a chunk. As a result, a malformed or malicious chunk that has `data_size = 0` will cause a panic in the `data_size_to_chunk_count` function, bringing down the entire process.

Furthermore, such an invalid chunk will already cause a panic earlier on in the handle_chunk_ingress_message function when processing a chunk, as it attempts to subtract `1` from `num_chunks_in_tx`, which is derived from `data_size.div_ceil(chunk_size)` . If `data_size` is 0, this will lead to a panic due to an arithmetic underflow.

```
// Is this chunk index any of the chunks before the last in the tx?
let num_chunks_in_tx = data_size.div_ceil(chunk_size);
if u64::from(*chunk.tx_offset) < num_chunks_in_tx - 1 {
    // Ensure prefix chunks are all exactly chunk_size
```

Impact

A user can intentionally send a chunk with `data_size = 0`, which will cause a panic during chunk processing, resulting in the entire process crashing and potentially leading to a denial of service.

Code Snippet

crates/database/src/db_cache.rs::data_size_to_chunk_count()

```
/// converts a size (in bytes) to the number of chunks, rounding up (size 0 -> illegal state, size 1 -> 1, size 262144 -> 1, 262145 -> 2 )
pub fn data_size_to_chunk_count(data_size: u64, chunk_size: u64) -> eyre::Result<u32> {
    assert_ne!(data_size, 0, "tx data_size 0 is illegal");
    Ok(data_size.div_ceil(chunk_size).try_into()?)
}
```

Tool used

Manual Review

Recommendation

Consider adding validation checks to ensure that `data_size` is greater than zero before processing a chunk.

Discussion

craigmayhew

https://github.com/Irys-xyz/irys/pull/620

Issue H-2: Merkle proof path validation panics due to arithmetic underflow if the path is smaller than `HASH_SIZE - NOTE_SIZE`

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/208

Summary

`validate_path()` panics due to an arithmetic underflow if the given `path_buff.len()` is < `(HASH_SIZE + NOTE_SIZE)`. This occurs when a user submits a chunk with an invalid (too small) `data_path` or if the gossiped block's `PoaData.tx_path` is too small.

Vulnerability Detail

`validate_path()` in `merkle.rs` panics due to an arithmetic underflow if the given `path_buff.len()` is < `(HASH_SIZE + NOTE_SIZE)`. This function is called during chunk ingress, allowing a user to purposefully cause a panic by providing an invalid (too small) `chunk.data_path`.

Additionally, `validate_path()` is also called in `poa_is_valid()` during block validation.

Impact

`validate_path()` panics due to an arithmetic underflow, causing the process to crash.

Code Snippet

crates/types/src/merkle.rs::validate_path()

```rust
pub fn validate_path(
    root_hash: [u8; HASH_SIZE],
    path_buff: &Base64,
    target_offset: u128,
) -> Result<ValidatePathResult, Error> {
    // Split proof into branches and leaf. Leaf is the final proof and branches
    // are ordered from root to leaf.
    let (branches, leaf) = path_buff.split_at(path_buff.len() - HASH_SIZE - NOTE_SIZE);
```

Tool used

Manual Review

Recommendation

Consider adding a check to ensure that the chunk's `data_path` length and `PoaData.tx_path` are correct and sufficiently long.

Discussion

antouhou

https://github.com/Irys-xyz/irys/pull/577

Issue H-3: Invalid chunk offset can cause node panic and crash

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/210

Summary

An invalid chunk offset in Proof of Access (PoA) data can cause the node to panic and crash due to an out-of-bounds array access in the block index lookup logic.

Vulnerability Detail

The vulnerability occurs in `get_block_index_item()` where a binary search can return an index equal to the array length, causing an immediate panic when accessing the array element:

```
// irys/crates/database/src/block_index_data.rs:172-184
pub fn get_block_index_item(&self, ledger: DataLedger, chunk_offset: u64) ->
↪ Result<(u64, &BlockIndexItem)> {
    let result = self.items.binary_search_by(|item| {
        if chunk_offset < item.ledgers[ledger as usize].max_chunk_offset {
            std::cmp::Ordering::Greater
        } else {
            std::cmp::Ordering::Less
        }
    });

    let index = match result {
        Ok(pos) => pos,
        Err(pos) => pos, // Can return items.len()
    };

    Ok((index as u64, &self.items[index])) // PANIC: index can be >= items.len()
}
```

When an attacker submits PoA data with a `ledger_chunk_offset` larger than any existing chunk offset, the binary search returns `items.len()`, causing `&self.items[index]` to panic with an out-of-bounds access.

Impact

- **Node Crash:** Immediate panic when processing malicious PoA data
- **Denial of Service:** Attackers can repeatedly crash nodes

- **Network Disruption:** Multiple node crashes could disrupt the network

Code Snippet

```
// irys/crates/database/src/block_index_data.rs:184
Ok((index as u64, &self.items[index])) // PANIC HERE
```

Tool Used

Manual Review

POC

You can test the panic by changing 150 to a value > 300 here: [https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/blob/681b311[...]46e451fbbb2dc50333/irys/crates/database/src/block_index_data.rs](https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/blob/681b311e968b5b48b89db446e451fbbb2dc50333/irys/crates/database/src/block_index_data.rs#L354)

Recommendation

Add bounds checking before array access:

```
pub fn get_block_index_item(&self, ledger: DataLedger, chunk_offset: u64) ->
 ↪ Result<(u64, &BlockIndexItem)> {
    // ... existing binary search code ...

    // Add bounds check
    if index >= self.items.len() {
        return Err(eyre::eyre!("Chunk offset {} exceeds maximum ledger size",
         ↪  chunk_offset));
    }

    Ok((index as u64, &self.items[index]))
}
```

Discussion

craigmayhew

Tracked by https://github.com/Irys-xyz/Irys-Internal/issues/27 and will be closed by https://github.com/Irys-xyz/irys/pull/589

Issue H-4: A malicious block containing `vdf_limiter_info.global_step_number = 0` results in a panic during block validation

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/213

Summary

A malicious block proposer can cause a denial of service attack by gossiping a block with `vdf_limiter_info.global_step_number = 0`, which results in an arithmetic underflow panic during block validation.

Vulnerability Detail

A gossiped and discovered block from a malicious block proposer containing `vdf_limiter_info.global_step_number = 0` results in an arithmetic underflow panic when calculating `(global_step_number - 1) as u64` in crates/vdf/src/lib.rs::last_step_checkpoints_is_valid()#L193 and in reset_step(), which is called from `recall_recall_range_is_valid()` as part of the the block pre-validation.

This can lead to a denial-of-service attack, as the block validation will panic, causing the process to crash and the node to stop processing blocks.

Impact

A malicious block proposer can cause a denial of service attack by gossiping such an invalid block that will panic when being processed by other peers, causing them to crash and stop processing blocks.

Code Snippet

crates/vdf/src/lib.rs#L193

```
190: // Calculate the starting salt value for checkpoint validation
191: let start_salt = U256::from(step_number_to_salt_number(
192:     config,
193:     (global_step_number - 1) as u64,
194: ));
```

crates/efficient-sampling/src/lib.rs#L168

```
pub fn reset_step(step_num: u64, num_recall_ranges_in_partition: u64) -> u64 {
    ((step_num - 1) / num_recall_ranges_in_partition) * num_recall_ranges_in_partition + 1
}
```

Tool used

Manual Review

Recommendation

Consider adding a check early in the block validation process to ensure that `vdf_limiter_info.global_step_number` is greater than 0.

Discussion

antouhou

https://github.com/Irys-xyz/irys/pull/579

Issue H-5: Incorrect logical operator allows invalid merkle proofs to bypass validation

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/218

Summary

The merkle proof validation logic uses an incorrect AND operator that allows invalid proofs to pass validation when targeting the root node, potentially compromising the integrity of the merkle tree verification system.

Vulnerability Detail

The validation check in the merkle proof verification contains a logical error:

```
// irys/crates/types/src/merkle.rs:291-293
if (id != root_id) && (data_hash != &leaf_proof.data_hash) { // @audit
    return Err(eyre!("Invalid Leaf Proof"));
}
```

The current logic only rejects proofs when **both** conditions are true:

1. `id != root_id` (not targeting the root node) **AND**
2. `data_hash != &leaf_proof.data_hash` (data hash mismatch)

This means when `id == root_id` (targeting the root), the first condition becomes `false`, causing the entire AND expression to evaluate to `false` regardless of whether the data hash matches. As a result, **any data hash is accepted when validating proofs targeting the root node**.

Attack Scenario:

1. Attacker crafts a merkle proof targeting the root node (`id == root_id`)
2. Provides an arbitrary/incorrect `data_hash` in the leaf proof
3. Validation passes because `(false) && (data_hash != &leaf_proof.data_hash) = false`
4. Invalid proof is accepted as valid

Impact

- **Merkle Tree Integrity Bypass:** Invalid proofs can be accepted when targeting root nodes

- **Data Authenticity Compromise:** Attackers can prove possession of data they don't actually have
- **Consensus Manipulation:** Invalid data could be accepted in consensus-critical operations
- **Storage Verification Failure:** Nodes may incorrectly validate non-existent or tampered data

Code Snippet

```
// irys/crates/types/src/merkle.rs:291-293
if (id != root_id) && (data_hash != &leaf_proof.data_hash) { // @audit incorrect AND
    return Err(eyre!("Invalid Leaf Proof"));
}
```

Tool Used

Manual Review

Recommendation

Use OR operator

```
if (id != root_id) || (data_hash != &leaf_proof.data_hash) {
    return Err(eyre!("Invalid Leaf Proof"));
}
```

Discussion

antouhou

The issue above has been fixed in one of the feature PRs we have merged since the snapshot

craigmayhew

Fixed in https://github.com/Irys-xyz/irys/pull/537

Issue H-6: Multiple unwrap calls without error handling can cause node crashes

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/220

Summary

The codebase contains numerous `unwrap()` calls without proper error handling that can cause immediate node panics and crashes during normal operation, particularly in critical consensus and validation paths.

Vulnerability Detail

Multiple critical components use `unwrap()` calls that can panic when encountering unexpected conditions:

Block Validation Critical Panics:

```
// irys/crates/actors/src/block_validation.rs:372
.unwrap(); //@audit error handling?

// irys/crates/actors/src/block_validation.rs:373
let ledger_chunk_offset = partition_assignment.slot_index.unwrap() as u64

// irys/crates/actors/src/block_validation.rs:380-381
let ledger = DataLedger::try_from(ledger_id).unwrap();//@audit error handling?
```

Database Transaction Panics:

```
// irys/crates/actors/src/block_tree_service.rs:136
.unwrap()

// irys/crates/actors/src/block_tree_service.rs:572
let tx = db.tx().unwrap();

// irys/crates/actors/src/mempool_service.rs:1022
let tx_ref = &reth_db.tx().unwrap();
```

Block Index Access Panics:

```
// irys/crates/database/src/block_index_data.rs:119
let prev_block = self.get_item(block.height.saturating_sub(1)).unwrap();

// irys/crates/database/src/block_index_data.rs:152
let previous_item = self.get_item(block_height - 1).unwrap();
```

```
// irys/crates/actors/src/block_tree_service.rs:254
let finalized = bi.get_item(finalized_height).unwrap();
```

Cache and Lock Panics:

```
// irys/crates/actors/src/block_tree_service.rs:48
self.block_tree_cache.read().unwrap()

// irys/crates/actors/src/block_tree_service.rs:372
let mut cache = binding.write().unwrap();

// irys/crates/actors/src/block_index_service.rs:30
self.block_index_data.read().unwrap()
```

Memory Allocation Panics:

```
// irys/crates/actors/src/mempool_service.rs:606
LruCache::new(NonZeroUsize::new(max_pending_pledge_items).unwrap());

// irys/crates/actors/src/mempool_service.rs:1561-1562
pending_chunks: LruCache::new(NonZeroUsize::new(max_pending_chunk_items).unwrap()),
pending_pledges:
 ↪  LruCache::new(NonZeroUsize::new(max_pending_pledge_items).unwrap()),
```

Consensus Chain Panics:

```
// irys/crates/actors/src/ema_service.rs:1328
let (mut latest_block_hash, ..) = *chain.last().unwrap();

// irys/crates/actors/src/mempool_service.rs:1497
let (latest_block_hash, _, _, _) = canonical_blocks.last().unwrap();

// irys/crates/actors/src/block_producer.rs:164
let (latest_block_hash, prev_block_height, _publish_tx, _submit_tx) =
 ↪  canonical_blocks.last().unwrap();
```

Impact

Any unwrap failure causes immediate panic and node shutdown

Tool Used

Manual Review

Recommendation

Replace all critical unwrap calls with proper error handling:

For Option types:

```
// Replace: partition_assignment.slot_index.unwrap()
let slot_index = partition_assignment.slot_index
    .ok_or_else(|| eyre::eyre!("Missing slot index in partition assignment"))?;
```

For Result types:

```
// Replace: DataLedger::try_from(ledger_id).unwrap()
let ledger = DataLedger::try_from(ledger_id)
    .map_err(|e| eyre::eyre!("Invalid ledger ID {}: {}", ledger_id, e))?;
```

For database operations:

```
// Replace: db.tx().unwrap()
let tx = db.tx()
    .map_err(|e| eyre::eyre!("Failed to create database transaction: {}", e))?;
```

For collection access:

```
// Replace: chain.last().unwrap()
let latest_block = chain.last()
    .ok_or_else(|| eyre::eyre!("Empty canonical chain"))?;
```

Implement a systematic review to replace all production unwrap calls with proper error handling that logs errors and gracefully handles failures.

Discussion

craigmayhew

Block Validation Critical Panics: https://github.com/Irys-xyz/irys/pull/564

Database Transaction Panics::

- Fixed: // irys/crates/actors/src/block_tree_service.rs:136
- Won't fix: // irys/crates/actors/src/block_tree_service.rs:572
- Fix this to be an Err() // irys/crates/actors/src/block_validation.rs:380-381

Block Index Access Panics::

- These examples need recovery code for the block index. i.e. read missing block header from the db instead of having an unwrap() error!

Cache and Lock Panics:: turn into expects() Memory Allocation Panics: turn into expects() Consensus Chain Panics: Some are in tests, change to expect(). Others are no longer present in codebase.

Issue H-7: Mempool transaction ingress signature validation bypass

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/224

Summary

The `handle_tx_ingress_message()` function in the mempool service is vulnerable to signature validation bypass as it ignores the result of the `validate_signature()` function.

Vulnerability Detail

`handle_tx_ingress_message()` handles an incoming data transaction and adds it to the mempool. It validates the transaction signature by calling `validate_signature()`, but ignores the result of the validation.

This means that if the signature is invalid, the transaction will still be added to the mempool and possibly included in a block. As a result, it allows forged transactions to be processed, allowing impersonation of other users and potentially leading to unauthorized actions.

Impact

Data transactions can be forged, and other users can be impersonated, leading to unauthorized actions.

Code Snippet

crates/actors/src/mempool_service.rs#L1200

```
1198: // Validate the transaction signature
1199: // check the result and error handle
1200: let _ = self.validate_signature(&tx).await;
```

Tool used

Manual Review

Recommendation

Consider handling the result of the `validate_signature()` call.

Discussion

craigmayhew

Fixed in master via https://github.com/Irys-xyz/irys/commit/cdd7fd56f26704a21d5b98b54b7baa76a69b8cac

Issue H-8: Forged peer connections via `/v1/version` endpoint

Source: https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/225

Summary

The `/v1/version` endpoint allows setting the `mining_address` without requiring a valid signature, enabling impersonation of peers and preventing nodes from establishing connections.

Vulnerability Detail

When initiating a new peer connection via the /v1/version endpoint, the `mining_address` can be set arbitrarily, which allows impersonating peers. This is caused by not requiring a valid signature for the handshake request, which would otherwise ensure that the `mining_address` is indeed controlled by the peer (i.e., the signer).

This vulnerability can be exploited by creating forged peer connections that appear to be from a legitimate peer, while the attacker will control the external listening address/port. As a result, the legitimate peer is unable to establish a connection with the nodes.

Impact

Nodes can be prevented from establishing connections with peers.

Tool used

Manual Review

Recommendation

Consider requiring a valid signature for the handshake request to ensure that the `mining_address` is controlled by the peer.

Discussion

antouhou

Fixed in https://github.com/Irys-xyz/irys/pull/546

Issue H-9: Arbitrary transaction `id`s and block header `block_hash` can be set

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/226

Summary

Transaction `id`s and block header `block_hash` can be set to arbitrary values, not derived from the hashed transaction signature.

Vulnerability Detail

The `id` of transactions (data and commitment transactions) and `block_hash` of block headers are supposed to be derived from the hashed transaction signature.

However, the corresponding is_signature_valid functions, that verify the signatures' validity, do not validate whether the `id` (and `block_hash`) is correctly derived. Consequently, arbitrary values can be provided.

Impact

For example, arbitrary transaction `id`s can be added to the `invalid_tx` mempool list, preventing the transaction from being included in a block. This can be used to grief other legitimate transactions.

And block headers can be created with arbitrary `block_hash` values that do not match the actual block content and signature, which can lead to inconsistencies in the blockchain state and consensus issues.

Tool used

Manual Review

Recommendation

Consider validating `id` and `block_hash` to ensure they are derived from the hashed signature.

Discussion

antouhou

Fixed in https://github.com/Irys-xyz/irys/pull/542

Issue H-10: Gossiped block cache pollution prevents legitimate blocks from being processed

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/229

Summary

Gossiped blocks will be recorded in the cache before validation, allowing forged blocks to prevent legitimate blocks with the same hash from being processed.

Vulnerability Detail

Whenever a gossiped block is processed via `handle_block_header_request()`, it records the seen blocked by caching the block hash.

However, this happens **before** the block validation. Therefore, it is possible to gossip forged blocks with a block hash that matches another legitimate block hash, but with a non-matching signature and different contents. As a result, the legitimate block, which is received later, has already been recorded before and will not be processed.

Impact

Forged blocks prevent the legitimate blocks with the same block hash from being processed, as they are already recorded in the cache.

Code Snippet

crates/p2p/src/server_data_handler.rs#L252-L267

```
226: pub(crate) async fn handle_block_header_request(
227:     &self,
228:     block_header_request: GossipRequest<IrysBlockHeader>,
229:     source_api_address: SocketAddr,
230: ) -> GossipResult<()> {
...      // [...]
251:
252:     let block_seen = self.cache.seen_block_from_any_peer(&block_hash)?;
253:
254:     // Record block in cache
255:     self.cache
256:         .record_seen(source_miner_address, GossipCacheKey::Block(block_hash))?;
257:
258:     // This check must be after we've added the block to the cache, otherwise
 ↪  we won't be
```

```
    // able to keep track of which peers seen what
    if block_seen {
        debug!(
            "Node {}: Block {} already seen, skipping",
            self.gossip_client.mining_address,
            block_header.block_hash.0.to_base58()
        );
        return Ok(());
    }
```

Tool used

Manual Review

Recommendation

Consider caching the block hash after the validation.

Discussion

antouhou

https://github.com/Irys-xyz/irys/pull/591

Issue H-11: Missing block validation checks

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/230

Summary

Certain block fields are not validated, which can lead to the acceptance of invalid blocks by the network.

Vulnerability Detail

Gossiped and received blocks from other peers are validated by prevalidate_block(). However, the validation lacks checks for the following fields:

- `last_diff_timestamp`
- `previous_solution_hash`
- `previous_cumulative_diff`
- `evm_block_hash`

Additionally, it also lacks validation that ensures the block height of a newly mined block is the parent height + 1. For example, if the current canonical chain tip height is 100, a gossiped block with the previous block at height 100, but with the new block's height set to 110 (skipping 10 blocks), will be accepted by the network.

Impact

The network can accept invalid blocks.

Tool used

Manual Review

Recommendation

Consider adding additional validation checks for discovered blocks.

Discussion

craigmayhew

https://github.com/Irys-xyz/irys/pull/606 - validate last_epoch_hash https://github.com/Irys-xyz/irys/pull/597 - validate last_diff_timestamp https://github.com/Irys-xyz/irys

/pull/595 - validate timestamp https://github.com/Irys-xyz/irys/pull/593 - validate previous_solution_hash https://github.com/Irys-xyz/irys/pull/618 - validate block height = parent block height + 1 https://github.com/Irys-xyz/irys/pull/617 - previous_cumulative_diff Won't fix: evm_block_hash is too heavy to compute in pre-validation but is checked in full block validation.

Issue H-12: Missing `chain_id` validation allows `IngressProof` reuse across networks

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/231

Summary

The protocol does not include or verify `chain_id` when signing and verifying `IngressProof`. This omission enables a malicious actor to reuse valid proofs from testnet on mainnet.

Vulnerability Detail

While generating and verifying `IngressProof`, the proof lacks chain-specific context – notably the `chain_id`. As a result, a miner or relayer could generate a valid proof on a testnet environment and replay it on mainnet. Since the proof doesn't carry the network identity, the system cannot differentiate between chains, leading to potential cross-network proof injection.

Impact

This flaw enables attackers to bypass the requirement of submitting data to the mainnet infrastructure. They can submit valid-looking proofs from a cheaper environment (e.g., testnet) without actually interacting with the mainnet. This compromises the integrity of state synchronization and could result in false state claims or DoS vectors.

Code Snippet

https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/blob/main/irys/crates/types/src/ingress.rs#L55-L64

Tool Used

Manual Review

Recommendation

Include `chain_id` as a required parameter in both signing and verifying the `IngressProof`. This ensures proofs are tightly bound to their origin network and cannot be replayed across chains.

Discussion

antouhou

https://github.com/Irys-xyz/irys/pull/583

Issue H-13: Missing validation in `Ranges::reinitialize` may lead to unconsumed range overwrites

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/233

Summary

The reinitialize function does not validate whether the previous range has been fully consumed (`last_range_pos == 0`) before allowing reinitialization. Since this function is public, it can be called at any time, risking data inconsistency.

Vulnerability Detail

The reinitialize function is callable externally and resets internal range-tracking state. However, it does not check if the previous range was fully consumed – typically by validating last_range_pos == 0. If this function is called while there are unconsumed ranges, the system may overwrite or discard in-progress data, leading to unexpected behavior or state corruption.

Impact

- Risk of data loss if existing ranges are overwritten without consumption
- Protocol may enter invalid or inconsistent state
- Potential denial-of-service vector if ranges are continually reset before completion

Code Snippet

https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/blob/main/irys/crates/efficient-sampling/src/lib.rs#L82-L97

Tool Used

Manual Review

Recommendation

Add a validation check at the start of reinitialize to ensure `last_range_pos == 0` before proceeding. Consider making the function internal or gated if external triggering poses risk to correctness.

Discussion

antouhou

I've submitted a fix, but I suspect that just check for 0 might be not enough, waiting for @JesseTheRobot to comment

https://github.com/Irys-xyz/irys/pull/578

antouhou

I've left a comment in the PR, but I'll duplicate it here just in case:

> We had a couple of discussions about this and came to a conclusion that re-initializing ranges that haven't been fully consumed is a part of normal operation. When syncing/receiving gossiped blocks mined by somebody else we don't consume any ranges. This results in the need for the range re-initialization when the local nodes try to mine a block after receiving a block mined by someone else

Issue H-14: Cross-Chain Replay Vulnerability in Commitment Transaction Creation

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/235

Summary

The `create_pledge_commitment_transaction` and genesis commitment creation functions in `system_ledger.rs` fail to set the `chain_id` field when creating `CommitmentTransaction` structs. This omission allows for cross-chain replay attacks where the same commitment transactions can be replayed across different blockchain networks.

Vulnerability Detail

In the create_pledge_commitment_transaction function and the genesis commitment creation, CommitmentTransaction structs are created using `..Default::default()` without explicitly setting the `chain_id` field. The `CommitmentTransaction` struct includes a `chain_id` field which is documented as being "used to prevent cross-chain replays".When `Default::default()` is used, the `chain_id` defaults to 0, making these transactions valid across all chains rather than being bound to a specific chain ID.

Impact

This vulnerability enables cross-chain replay attacks where:

1. An attacker can capture commitment transactions from one network
2. Replay them on another network with a different chain ID
3. Potentially cause unauthorized staking/pledging operations or disrupt network consensus
4. Compromise the integrity of the multi-chain deployment by allowing commitments intended for one chain to affect another

Code Snippet

https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/blob/main/irys/crates/database/src/system_ledger.rs#L103-L107
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/blob/main/irys/crates/database/src/system_ledger.rs#L150-L154

Tool Used

Manual Review

Recommendation

Explicitly set the chain_id field in all `CommitmentTransaction` creations

Discussion

antouhou

Has been fixed by @roberts-pumpurs in an unrelated feature PR:
https://github.com/Irys-xyz/irys/pull/548/files
allowbreak
#diff-56f0672ee4b3ca39094530c95e67d9e16fdec93ef9b9c6c14c444c8c2e3d32ecR109

Issue H-15: Weak Commitment Validation Allows Processing of Extra Transactions

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/237

Summary

The `validate_commitments()` function in `EpochServiceActor` performs insufficient validation of commitment transactions during epoch initialization and replay scenarios. While the function verifies that all commitment IDs referenced in the block's ledger exist in the provided commitments vector, it fails to validate that no extra commitment transactions are included. This allows additional commitments beyond those referenced in the block to be processed and stored in the commitment state.

Vulnerability Detail

The validation logic only performs a one-way check ensuring every transaction ID in the block's commitment ledger has a corresponding transaction in the commitments vector. The function does not validate that `commitment_ledger.tx_ids.len() == commitments.len()`, allowing scenarios where extra commitment transactions are present in the commitments vector. These extra commitments bypass validation and get processed by `compute_commitment_state()`, even though they're not part of the actual epoch block.

Impact

This vulnerability enables attackers to inject unauthorized commitment transactions that bypass validation and corrupt the network's mining consensus mechanism. Extra commitments processed by `compute_commitment_state()` create invalid stake and pledge entries in the commitment state, which then get assigned partition hashes through `assign_partition_hashes_to_pledges()`, effectively granting mining rights to addresses that made no legitimate on-chain commitments. This breaks the fundamental proof-of-commitment security model by allowing attackers to gain economic value through unauthorized mining assignments.

Code Snippet

https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/blob/main/irys/crates/actors/src/epoch_service/epoch_service.rs#L150-L170

Tool Used

Manual Review

Recommendation

Implement strict bidirectional validation to ensure exact matching between the block's commitment ledger and provided commitments

Discussion

antouhou

https://github.com/Irys-xyz/irys/pull/582

Issue H-16: Integer Underflow DoS Vulnerability in Merkle Path Validation

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/238

Summary

A critical integer underflow vulnerability exists in the `validate_path` function that allows external attackers to crash Irys nodes by sending malformed merkle proofs. When `path_buff.len() < 64`, the subtraction `path_buff.len() - HASH_SIZE - NOTE_SIZE` causes integer underflow, leading to a panic and node crash.

Vulnerability Detail

When `path_buff.len() < 64` (HASH_SIZE + NOTE_SIZE), the subtraction underflows, causing panic and crash the node.

Impact

Code Snippet

https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/blob/main/irys/crates/types/src/merkle.rs#L123

Tool Used

Manual Review

Recommendation

Add bounds check: `if path_buff.len() < HASH_SIZE + NOTE_SIZE { return Err(...) }` before the split operation.

Discussion

antouhou

https://github.com/Irys-xyz/irys/pull/577

Issue M-1: Missing to prune chunks and ingress proofs for data roots that never had all chunks received

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/206

Summary

Chunks and ingress proofs for data roots that never received all chunks are never pruned, leading to indefinite storage occupation on the node.

Vulnerability Detail

prune_data_root_cache() only prunes chunks and ingress proofs if there is a corresponding `DataRootLRUEntry` entry. However, this LRU entry is only added once all chunks for a data root have been received.

If not all chunks are ever received for a data root, it indefinitely occupies storage on the node, which could be used to spam the node and fill up storage, eventually leading to a denial of service.

Impact

Nodes eventually run out of storage due to unpruned data roots that never received all their chunks, leading to a denial of service.

Code Snippet

crates/actors/src/cache_service.rs::prune_data_root_cache()

```
fn prune_data_root_cache(&self, prune_height: u64) -> eyre::Result<()> {
    let mut chunks_pruned: u64 = 0;
    let write_tx = self.db.tx_mut()?;
    let mut cursor = write_tx.cursor_write::<DataRootLRU>()?;
    let mut walker = cursor.walk(None)?;
    while let Some((data_root, DataRootLRUEntry { last_height, .. })) =
        walker.next().transpose()?
    {
        if last_height < prune_height {
            debug!(
                ?data_root,
                ?last_height,
                ?prune_height,
                "expiring ingress proof",
            );
            write_tx.delete::<DataRootLRU>(data_root, None)?;
            write_tx.delete::<IngressProofs>(data_root, None)?;
            // delete the cached chunks
            chunks_pruned = chunks_pruned
                .saturating_add(delete_cached_chunks_by_data_root(&write_tx, data_root)?);
        }
    }
    debug!(?chunks_pruned, "Pruned chunks");
    write_tx.commit()?;

    Ok(())
}
```

Tool used

Manual Review

Recommendation

Consider expiring ”incomplete” data roots after a certain period.

Discussion

craigmayhew

https://github.com/Irys-xyz/irys/pull/638

Issue M-2: Commitment transactions are not checked for sufficient funding

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/211

Summary

Commitment transactions are not checked for sufficient funding, resulting in indefinitely growing memory usage of the nodes due to unprocessed transactions that cannot be included in a block.

Vulnerability Detail

During block building, `handle_get_best_mempool_txs()` checks the balance of the transaction signers to ensure they have sufficient funds to cover the transaction fees. Contrary to handle_tx_ingress_message(), which processes data transactions, handle_commitment_tx_ingress_message() does not perform a balance check for new commitment transactions.

This means that the mempool can be filled with commitment transactions from signers who do not have enough balance to cover the total fee. As a result, these transactions may never be included in a block, leading to increased memory usage, wasting the nodes' resources, and potentially causing a denial-of-service.

Impact

Increased memory usage of the nodes due to unprocessed commitment transactions that cannot be included in a block because the signers do not have sufficient funds to cover the transaction fees.

Tool used

Manual Review

Recommendation

Consider adding a balance check for new commitment transactions in the `handle_commitment_tx_ingress_message()` and using LRU caching to manage pending transactions per signer more effectively.

Issue M-3: Missing signature validation for `Unstaked` commitment transactions

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/212

Summary

`handle_commitment_tx_ingress_message()`, which processes newly submitted commitment transactions as part of the mempool service, does not validate the signature of `Unstaked` commitment transactions before adding them to the LRU cache. This can lead to arbitrary transactions being added, potentially evicting legitimate transactions from the cache.

Vulnerability Detail

In `handle_commitment_tx_ingress_message()`, the signature validation is missing for Unstaked commitments, resulting in arbitrary commitment transactions being added to the LRU cache. This can be misused to add a large number of transactions to the LRU cache for a specific signer address, evicting other legitimate transactions from the cache so that they will no longer be processed.

Notably, such unvalidated transactions will have their signature validated once they are re-processed and accepted.

Impact

Legitimate transactions can be evicted from the LRU cache for specific and arbitrary signer addresses, preventing them from being processed and potentially leading to a denial-of-service attack.

Code Snippet

crates/actors/src/mempool_service.rs#L590-L617

```
590: if commitment_status == CommitmentCacheStatus::Unstaked {
591:     // For unstaked pledges, we cache them in a 2-level LRU structure:
592:     // Level 1: Keyed by signer address (allows tracking multiple addresses)
593:     // Level 2: Keyed by transaction ID (allows tracking multiple pledge tx per address)
594:
595:     let mut mempool_state_guard = mempool_state.write().await;
596:     if let Some(pledges_cache) = mempool_state_guard
597:         .pending_pledges
```

```
        .get_mut(&commitment_tx.signer)
    {
        // Address already exists in cache - add this pledge transaction to its lru cache
        pledges_cache.put(commitment_tx.id, commitment_tx.clone());
    } else {
        // First pledge from this address - create a new nested lru cache
        let max_pending_pledge_items =
            self.config.consensus.mempool.max_pending_pledge_items;
        let mut new_address_cache =
            LruCache::new(NonZeroUsize::new(max_pending_pledge_items).unwrap());

        // Add the pledge transaction to the new lru cache for the address
        new_address_cache.put(commitment_tx.id, commitment_tx.clone());

        // Add the address cache to the primary lru cache
        mempool_state_guard
            .pending_pledges
            .put(commitment_tx.signer, new_address_cache);
    }
    drop(mempool_state_guard)
} else {
```

Tool used

Manual Review

Recommendation

Consider adding signature validation for `Unstaked` commitments in the `handle_commitment_tx_ingress_message()` function to ensure that only valid transactions are added to the LRU cache.

Discussion

craigmayhew

Fixed in https://github.com/Irys-xyz/irys/pull/567

Issue M-4: Abi encoded data conversion failure causes default values

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/215

Summary

The `try_into()` conversion in system transaction encoding can fail when the ABI-encoded data size doesn't match the expected fixed-size array, causing the function to return default values instead of the actual encoded data.

Vulnerability Detail

In the system transaction encoding logic, `abi_encode_packed()` returns variable-length data that is then converted to a fixed-size type using `try_into()`:

```
// irys/crates/irys-reth/src/system_tx.rs:88-97
Self::ReleaseStake(bi) | Self::BlockReward(bi) => {
    use alloy_dyn_abi::DynSolValue;
    DynSolValue::Tuple(vec![
        DynSolValue::Uint(bi.amount, 256),
        DynSolValue::Address(bi.target),
    ])
    .abi_encode_packed()
    .try_into() // @audit will fail
    .unwrap_or_default()
}
```

When the encoded tuple size doesn't match the expected fixed-size array length, `try_into()` fails and the function returns default values (likely zeros) instead of the actual encoded stake release or block reward data.

Code Snippet

```
// irys/crates/irys-reth/src/system_tx.rs:95-97
.abi_encode_packed()
.try_into() // @audit will fail
.unwrap_or_default()
```

POC

```
fn test_poc() {
        let packet = TransactionPacket::ReleaseStake(BalanceIncrement {
            amount: uint!(1000_U256),
            target: address!("1234567890123456789012345678901234567890"),
        });

        let topic = packet.encoded_topic();

        assert_eq!(topic, [0u8; 32]);
    }
```

Issue M-5: Incorrect loop control slows down block discovery consensus

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/216

Summary

Using `continue` instead of `break` in peer block discovery causes unnecessary retries on peers that don't have the requested block, significantly slowing down consensus performance.

Vulnerability Detail

The peer discovery logic incorrectly uses `continue` when it should use `break`:

```
// irys/crates/p2p/src/peer_list.rs:1317
continue; //@audit
```

This causes the system to retry the same peer 5 times even when it's confirmed the peer doesn't have the block, instead of moving to the next peer immediately. The current behavior performs 5 iterations per peer sequentially, wasting time on peers that cannot provide the requested block.

Impact

- **Consensus Delay**: Block discovery takes significantly longer due to unnecessary retries
- **Network Inefficiency**: Wasted bandwidth and CPU cycles on futile retry attempts
- **Performance Degradation**: Each peer gets 4 extra unnecessary requests before moving to the next peer
- **Reduced Throughput**: Overall network performance suffers from suboptimal peer selection

Code Snippet

```
// irys/crates/p2p/src/peer_list.rs:1317
continue; //@audit should be break
```

Tool Used

Manual Review

Recommendation

1. **Fix the immediate bug**:

```
// Replace continue with break
break; // Move to next peer immediately
```

2. **Optimize the retry strategy**:

```
// Instead of 5 iterations per peer, iterate each peer once and loop 5 times
for attempt in 0..5 {
    for peer in &peers {
        if let Some(block) = try_get_block_from_peer(peer) {
            return Some(block);
        }
    }
    // Brief delay between full peer list iterations
    tokio::time::sleep(Duration::from_millis(100)).await;
}
```

This approach distributes requests more evenly across peers and reduces overall discovery time.

Discussion

craigmayhew

https://github.com/Irys-xyz/irys/pull/644

Issue M-6: incorrect buffer length checks may cause valid addresses to be rejected

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/217

Summary

The `decode_address` function has incorrect buffer length checks that cause valid IPv4 and IPv6 addresses to be rejected and replaced with default values, leading to connection failures and network communication issues.

Vulnerability Detail

The buffer length validation in `decode_address` uses incorrect thresholds:

```rust
// irys/crates/types/src/peer_list.rs:130-147
fn decode_address(buf: &[u8]) -> (SocketAddr, usize) {
    let tag = buf[0];
    let address = match tag {
        0 => {
            // IPv4 address (needs 4 bytes IP + 2 bytes port after tag)
            if buf.len() < 11 { //@audit should be < 7
                SocketAddr::V4(SocketAddrV4::new(Ipv4Addr::new(0, 0, 0, 0), 0))
            } else {
                // ... valid IPv4 decoding
            }
        }
        1 => {
            // IPv6 address (needs 16 bytes IP + 2 bytes port after tag)
            if buf.len() < 23 { //@audit should be < 19
                SocketAddr::V4(SocketAddrV4::new(Ipv4Addr::new(0, 0, 0, 0), 0))
            } else {
                // ... valid IPv6 decoding
            }
        }
    }
}
```

Actual Requirements:

- IPv4: 1 byte tag + 4 bytes IP + 2 bytes port = **7 bytes total**
- IPv6: 1 byte tag + 16 bytes IP + 2 bytes port = **19 bytes total**

Current Incorrect Checks:

- IPv4: Requires 11 bytes (4 bytes too many)

- IPv6: Requires 23 bytes (4 bytes too many)

This causes valid addresses with correct minimal buffers to be rejected and replaced with default `0.0.0.0:0` addresses.

Impact

- **Connection Failures:** Valid peer addresses are replaced with invalid default addresses
- **Network Partitioning:** Nodes cannot connect to peers with correctly encoded addresses
- **P2P Network Degradation:** Reduced peer connectivity affects network robustness
- **Protocol Incompatibility:** Peers using minimal correct encoding cannot communicate

Code Snippet

```rust
// Incorrect length checks
if buf.len() < 11 { // Should be < 7 for IPv4
if buf.len() < 23 { // Should be < 19 for IPv6
```

Proof of Concept

```rust
// irys/crates/types/src/peer_list.rs:324-335
#[test]
fn PoC() {
    let original_address = SocketAddr::V4(SocketAddrV4::new(Ipv4Addr::new(100, 0,
     ↪  0, 1), 2000));

    let mut buf = bytes::BytesMut::with_capacity(10);
    let size = encode_address(&original_address, &mut buf);
    assert_eq!(size, 7);
    assert_eq!(buf.len(), 7);

    let (decoded_address, consumed) = decode_address(&buf[..]);
    assert_eq!(consumed, 7);
    assert_eq!(decoded_address, original_address); // This will fail with current
     ↪  bug
}
```

This test demonstrates that IPv4 addresses encode to exactly 7 bytes, but the current length check `< 11` would incorrectly reject this valid 7-byte buffer and return `0.0.0.0:0` instead of the original address `100.0.0.1:2000`.

Tool Used

Manual Review

Recommendation

Fix the buffer length checks to use the correct minimum sizes:

```rust
fn decode_address(buf: &[u8]) -> (SocketAddr, usize) {
    let tag = buf[0];
    let address = match tag {
        0 => {
            // IPv4: 1 tag + 4 IP + 2 port = 7 bytes total
            if buf.len() < 7 {
                SocketAddr::V4(SocketAddrV4::new(Ipv4Addr::new(0, 0, 0, 0), 0))
            } else {
                // ... existing IPv4 decoding logic
            }
        }
        1 => {
            // IPv6: 1 tag + 16 IP + 2 port = 19 bytes total
            if buf.len() < 19 {
                SocketAddr::V4(SocketAddrV4::new(Ipv4Addr::new(0, 0, 0, 0), 0))
            } else {
                // ... existing IPv6 decoding logic
            }
        }
        _ => SocketAddr::V4(SocketAddrV4::new(Ipv4Addr::new(0, 0, 0, 0), 0)),
    };
    // ... rest of function
}
```

Issue M-7: Off-by-one error in `ledger_chunk_offset_ie` macro by using `ii` instead of `ie`

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/221

Summary

The `ledger_chunk_offset_ie` Rust macro returns an **inclusive-inclusive** range instead of an **inclusive-exclusive** range, leading to overlapping storage slot ranges.

Vulnerability Detail

The `ledger_chunk_offset_ie` Rust macro is supposed to return an **inclusive-exclusive** range. However, it uses `ii`, which returns an inclusive-inclusive range, manifesting as an off-by-one error.

This leads to the creation of ranges with overlapping storage slots, causing issues in the `get_storage_module_ledger_range()` caller function and potentially resulting in incorrect data retrieval or storage operations.

Impact

Storage slot ranges will overlap, leading to potential data corruption or retrieval issues.

Code Snippet

crates/types/src/storage.rs#L330-L337

```
329: #[macro_export]
330: macro_rules! ledger_chunk_offset_ie {
331:     ($start:expr, $end:expr) => {
332:         ii(
333:             LedgerChunkOffset::from($start),
334:             LedgerChunkOffset::from($end),
335:         )
336:     };
337: }
```

Tool used

Manual Review

Recommendation

Consider using `ie` instead of `ii` in the `ledger_chunk_offset_ie` macro.

Discussion

craigmayhew

closed by https://github.com/Irys-xyz/irys/pull/581

Issue M-8: Mempool pending chunks LRU cache cache pollution

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/222

Summary

The `handle_chunk_ingress_message()` function in the mempool service is vulnerable to cache pollution attacks as it allows adding chunks to the pending chunks LRU cache without validating their integrity.

Vulnerability Detail

In `handle_chunk_ingress_message()`, chunks are added to the pending chunks LRU cache for the given `data_root`, without validating whether the chunks belong to the root.

This allows adding bogus chunks for a valid data root, so that other valid chunks are kicked out of the LRU cache due to reaching capacity.

Impact

The pending chunks LRU cache can be filled with bogus chunks, leading to valid chunks being evicted from the cache and causing a denial of service for the specific data root.

Code Snippet

crates/actors/src/mempool_service.rs#L780-L782

```
762: let data_size = match data_size {
763:     Some(ds) => ds,
764:     None => {
765:         let mut mempool_state_write_guard = mempool_state.write().await;
766:         // We don't have a data_root for this chunk but possibly the transaction containing this
767:         // chunks data_root will arrive soon. Park it in the pending chunks LRU cache until it does.
768:         if let Some(chunks_map) = mempool_state_write_guard
769:             .pending_chunks
770:             .get_mut(&chunk.data_root)
771:         {
772:             chunks_map.put(chunk.tx_offset, chunk.clone());
773:         } else {
774:             // If there's no entry for this data_root yet, create one
775:             let mut new_lru_cache = LruCache::new(
                NonZeroUsize::new(max_chunks_per_item)
                    .expect("expected valid NonZeroUsize::new"),
            );
            new_lru_cache.put(chunk.tx_offset, chunk.clone());
            mempool_state_write_guard
                .pending_chunks
                .put(chunk.data_root, new_lru_cache);
        }
        drop(mempool_state_write_guard);
        return Ok(());
    }
};
```

Tool used

Manual Review

Recommendation

Consider validating the chunk and its path before adding it to the pending chunks LRU.

Discussion

craigmayhew

https://github.com/Irys-xyz/irys/pull/615

Issue L-1: Duplicate condition and lack of inclusive comparison in block finalization check

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/204

Summary

`try_notify_services_of_block_finalization()` contains a duplicate condition and uses > instead of >= when checking if a block is finalized, which can lead to an already finalized block being re-finalized.

Vulnerability Detail

In the try_notify_services_of_block_finalization() function, it checks if the block is already finalized:

```
if bi.num_blocks() > finalized_height && bi.num_blocks() > finalized_height {
    let finalized = bi.get_item(finalized_height).unwrap();
    if finalized.block_hash == finalized_hash {
        return;
    }
    panic!("Block tree and index out of sync");
}
```

First, the condition is repeated. Second, the comparison operator is >, which means that if the index contains `finalized_height` blocks, it will not check the block at `finalized_height`. This is because `bi.num_blocks() > finalized_height` will be `false` when `bi.num_blocks()` is equal to `finalized_height`.

Instead, it should use `bi.num_blocks() >= finalized_height` to ensure that the block at `finalized_height` is included in the check.

Impact

An already finalized block can potentially be re-finalized.

Tool used

Manual Review

Recommendation

Consider removing the duplicate condition and using >= instead of > to ensure that the block at `finalized_height` is checked.

Discussion

craigmayhew

https://github.com/Irys-xyz/irys/pull/633

Issue L-2: Chunks can be spammed without paying fees before the data transaction is included in a block

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/205

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

Summary

Ingress chunks get cached immediately before the corresponding data transaction is included in a block, before the fee is paid. This allows spamming nodes with chunks that fill up memory and storage without any cost.

Vulnerability Detail

The data root is immediately cached during handle_data_tx_ingress_message() when processing a newly submitted data transaction, before the transaction is even included in the block (and the fee actually paid).

Consequently, incoming chunks bypass the LRU cache due to `data_size` being `Some(..)` and get immediately cached via `irys_database::cache_chunk()`.

Additionally, an ingress proof is generated once all chunks are received and the associated chunks are all gossiped to other peers, putting pressure on the network, even though the data transaction is not yet included in the block and may never be included.

Essentially, this allows spamming the nodes with chunks that fill up memory and storage.

Impact

Nodes can be spammed with chunks that fill up memory and storage.

Code Snippet

crates/actors/src/mempool_service.rs#L1209

```
1207: // Cache the data_root in the database
1208: match self.irys_db.update_eyre(|db_tx| {
1209:     irys_database::cache_data_root(db_tx, &tx)?;
```

Tool used

Manual Review

Recommendation

Consider adding further constraints to prevent large-scale chunk spamming.

Discussion

DanMacDonald

In this case the `pending_chunks` is an LRU of LRUs. The internal LRU is an LRU per `data_root` for `data_roots` that have not yet been confirmed by being part of a valid data transaction received by the node.

The way chunks get into this double LRU (which constrains 1. how many unconfirmed data_roots can be tracked and 2. how many chunks to hold per unconfirmed `data_root`, is by there not being a valid transaction that confirms the chunks present on the node.

This has a few consequences.

1. The number of unconfirmed data_roots to track is limited by the outer LRU
2. The number of chunks per unconfirmed data_root is limited by the inner LRU

^ This means the consequence of any kind of spamming is to just cycle/flush these LRUs.

The above is a low impact strategy because as soon as the valid TX with the `data_root` is confirmed (included in a block in the canonical chain) the above logic is skipped and the chunks are processed normally. In the `Some(ds)` path.

```
let data_size = match data_size {
           Some(ds) => ds,
```

DanMacDonald

These "optimistic" pending chunk LRUs are just an optimization, if an adversary was to attack them they might degrade performance slightly but they would not interrupt the function of the node.

lpetroulakis

After the discussion with the team, there is no need for an immediate fix here. Downgrading this to `Info`. The issue is acknowledged as a potential UX one, but the impact is minimal.

Issue L-3: Off-by-One error in `BlockTreeCache` `prune()` function

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/209

Summary

The `prune` function in `BlockTreeCache` contains an off-by-one error in the calculation of `min_keep_height`, causing it to retain one more block than intended when pruning the block tree cache.

Vulnerability Detail

In the `prune` function at line 1192, the minimum height threshold for keeping blocks is calculated as:

```
let min_keep_height: u64 = tip_height.saturating_sub(depth);
```

However, this calculation results in keeping `depth + 1` blocks instead of the intended `depth` blocks. For example:

- If `tip_height = 10` and `depth = 3` (intending to keep 3 blocks from tip)
- `min_keep_height = 10 - 3 = 7`
- The loop prunes blocks where `current_height < 7` (heights 0-6)
- This keeps blocks at heights 7, 8, 9, 10 (4 blocks instead of 3)

The loop condition `while current_height < min_keep_height` removes all blocks below the calculated threshold, but the threshold itself is off by one.

Impact

Currently the function isn't used in the codebase, however, if used in the future, it cause incorrect number of blocks to be stored in cache than intended.

Code Snippet

```
// irys/crates/actors/src/block_tree_service.rs:1192
let min_keep_height: u64 = tip_height.saturating_sub(depth);// @audit off by one
```

Tool Used

Manual Review

Recommendation

Fix the off-by-one error by adjusting the `min_keep_height` calculation:

```
let min_keep_height: u64 = tip_height.saturating_sub(depth).saturating_add(1);
```

Issue L-4: Incorrect chunk offset validation

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/219

Description

The chunk offset validation uses an inclusive range check (`..=`) that conflicts with the exclusive binary search logic. When `ledger_chunk_offset` equals `max_chunk_offset`, the binary search returns the next block index (potentially out-of-bounds), but the inclusive range check passes validation.

Recommendation

Make the range check exclusive to match the binary search semantics:

```
// irys/crates/actors/src/block_validation.rs:385-387
if !(bb.start_chunk_offset..bb.end_chunk_offset).contains(&ledger_chunk_offset) {
    return Err(eyre::eyre!("PoA chunk offset out of block bounds"));
};
```

Discussion

craigmayhew

https://github.com/Irys-xyz/irys/pull/636

Issue L-5: Inconsistent known peer address cache update by re-inserting old address

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/223

Summary

`update_peer_address()` incorrectly updates the `known_peers_cache` by re-inserting the old peer address instead of the new one, leading to inconsistencies in the cache.

Vulnerability Detail

The `update_peer_address` function updates the external listening address (`PeerAddress`) for the given `mining_addr`. However, the known peers cache is incorrectly updated by re-inserting the previous peer address instead of the new one. This leads to inconsistencies in the cache.

Impact

The `known_peers_cache` becomes inconsistent as it retains old peer addresses.

Code Snippet

crates/p2p/src/peer_list.rs#L562

```
553: fn update_peer_address(&mut self, mining_addr: Address, new_address:
 ↪  PeerAddress) {
554:     if let Some(peer) = self.peer_list_cache.get_mut(&mining_addr) {
555:         let old_address = peer.address;
556:         peer.address = new_address;
557:         self.gossip_addr_to_mining_addr_map
558:             .remove(&old_address.gossip.ip());
559:         self.gossip_addr_to_mining_addr_map
560:             .insert(new_address.gossip.ip(), mining_addr);
561:         self.known_peers_cache.remove(&old_address);
562:         self.known_peers_cache.insert(old_address);
563:         self.api_addr_to_mining_addr_map.remove(&old_address.api);
564:         self.api_addr_to_mining_addr_map
565:             .insert(new_address.api, mining_addr);
566:     }
567: }
```

Tool used

Manual Review

Recommendation

Consider inserting the new address, `new_address`, into the `known_peers_cache` instead of the old address, `old_address`, to maintain consistency in the cache.

Issue L-6: Unbounded `limit` parameter in `/v1/block_index` REST API endpoint

Source: https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/227

Summary

The `limit` parameter in the `/v1/block_index` REST API endpoint has no maximum limit, which could lead to performance issues or denial of service if a large range is requested.

Vulnerability Detail

`block_index_route()` handles the `/v1/block_index` REST API endpoint and allows querying a range of blocks based on the `height` and `limit` parameters.

However, it does not impose a maximum limit on the number of blocks that can be queried, which could lead to performance issues or denial of service if a large range is requested.

Impact

Potential performance issues or denial of service due to unbounded queries on the block index.

Code Snippet

crates/api-server/src/routes/block_index.rs::block_index_route()

```
pub async fn block_index_route(
    state: web::Data<ApiState>,
    query: web::Query<BlockIndexQuery>,
) -> HttpResponse {
    let limit = query.limit;
    let height = query.height;

    let block_index_read = state.block_index.read();
    let requested_blocks: Vec<&BlockIndexItem> = block_index_read
        .items
        .into_iter()
        .enumerate()
        .filter(|(i, _)| *i >= height && *i < height + limit)
        .map(|(_, block)| block)
        .collect();

    HttpResponse::Ok()
        .content_type(ContentType::json())
        .body(serde_json::to_string_pretty(&requested_blocks).unwrap())
}
```

Tool used

Manual Review

Recommendation

Consider imposing a maximum limit on the `limit` parameter.

Discussion

craigmayhew

https://github.com/Irys-xyz/irys/pull/634

Issue L-7: Overflow checks should be enabled in production builds

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/228

Summary

The `overflow-checks` feature is not enabled in `Cargo.toml`, which is particularly important for production builds as it ensures that arithmetic operations are checked and panics to prevent silent integer overflows.

Vulnerability Detail

The `Cargo.toml` configuration files do not specify the `overflow-checks` feature, which is required to enable overflow checks in the Rust compiler. This can lead to potential silent integer overflows in arithmetic operations, particularly when handling large numbers or calculations that exceed the maximum value of the data type.

Impact

Without overflow checks enabled in production, the application may experience unexpected behavior due to integer overflows.

Tool used

Manual Review

Recommendation

Consider adding `overflow-checks = true` to the `Cargo.toml` files to enable overflow checks in production builds.

Discussion

craigmayhew

@JesseTheRobot

Issue L-8: Redundant Commitment Validation in `EpochService` Initialization

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/236

Summary

The `initialize()` method in `EpochServiceActor` contains a redundant call to `validate_commitments()`. The same validation is performed twice in the same execution path - once explicitly in `initialize()` and again within the subsequent call to `perform_epoch_tasks()`.

Vulnerability Detail

In the `initialize()` function, `validate_commitments()` is called explicitly before calling `perform_epoch_tasks()`. However, `perform_epoch_tasks()` performs the exact same validation. This creates redundant validation logic.

Impact

Code redundancy

Code Snippet

https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/blob/main/irys/crates/actors/src/epoch_service/epoch_service.rs#L101

Tool Used

Manual Review

Recommendation

Remove the redundant code

Discussion

craigmayhew

@craigmayhew todo

craigmayhew

https://github.com/Irys-xyz/irys/pull/627

Issue L-9: Incorrect Byte Range Assignment in Merkle Branch Construction

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/239

Summary

Two bugs in `hash_branch()` function in merkle.rs:

1. Branch hash calculation uses `left.max_byte_range.to_note_vec()` but should use `right.max_byte_range` since branch nodes represent the maximum range of their children
2. Node construction sets `min_byte_range: left.max_byte_range` but should be `min_byte_range: left.min_byte_range` to properly represent the minimum range

Impact

Corrupts merkle tree structure leading to invalid proofs, failed block validation, and potential consensus failures as nodes may disagree on valid merkle roots.

Code Snippet

https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/blob/main/irys/crates/types/src/merkle.rs#L415 https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/blob/main/irys/crates/types/src/merkle.rs#L420

Tool Used

Manual Review

Recommendation

Fix both assignments: use `right.max_byte_range.to_note_vec()` for hashing and `left.min_byte_range` for the min_byte_range field.

Discussion

craigmayhew

~@craigmayhew todo~ Done https://github.com/Irys-xyz/irys/pull/630

Issue L-10: Integer overflow in cache size calculation

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/240

Description

The `get_cache_size` function performs unchecked multiplication that can overflow when `chunk_count` is large, potentially causing incorrect cache size calculations or wraparound behavior.

```
// irys/crates/database/src/database.rs:271
Ok((chunk_count as u64, chunk_count as u64 * chunk_size)) // @audit safe?
```

Recommendation

Use `checked_mul()` to safely handle potential overflow:

```
// irys/crates/database/src/database.rs:271
let total_size = (chunk_count as u64).checked_mul(chunk_size)
    .ok_or_else(|| eyre::eyre!("Cache size calculation overflow"))?;
Ok((chunk_count as u64, total_size))
```

Discussion

craigmayhew

~@craigmayhew todo~ Done https://github.com/Irys-xyz/irys/pull/629

Issue L-11: Node panic from missing chunk data

Source:
https://github.com/sherlock-audit/2025-06-irys-mainnet-audit-june-2nd/issues/241

Description

The database code uses `.expect()` which can cause immediate node crash when a chunk has an index entry but missing data entry, indicating a database inconsistency state that could occur during concurrent operations or corruption.

```
// irys/crates/database/src/database.rs:235-236
.expect("Chunk has an index entry but no data entry"),
```

Recommendation

Replace `.expect()` with proper error handling to prevent node crashes:

```
// irys/crates/database/src/database.rs:235-236
.ok_or_else(|| eyre::eyre!("Chunk has an index entry but no data entry"))?
```

Discussion

craigmayhew

@craigmayhew this should be an error not an expect/panic

Disclaimers