QUORINT

SECURITY ASSESSMENT

Zakura

Zcash validator full node. Consensus, networking, state and RPC.

35 FINDINGS, IN REPORT ORDER



Critical **1** High **1** Medium **13** Low **12** Informational **8**

PREPARED FOR	PREPARED BY	ENGAGEMENT
Dev Ojha (Zakura)	Ouicate	Full-node security review
REVIEWED	**PERIOD & DATE**	**VERSION**
`v1.6.0` (`ebda15e23`), `main` (`38af6a6a` / `454f0767`), recheck `v1.7.0-rc0` (`cd103f69`)	2 Oct - 13 Oct 2026	1.0

© 2026 Quorint. All rights reserved.

quorint.xyz contact@quorint.xyz

Contents

1 Summary **3**
2 What we reviewed **4**
2.1 The code 4
2.2 What we did not cover 4
2.3 Timeline and people 4
3 How we worked **5**
3.1 What we set out to answer 5
3.2 Method 5
3.3 How we rate 5
4 Findings at a glance **6**
5 Findings **8**
Critical (1) 8
High (1) 10
Medium (13) 11
Low (12) 23
Informational (8) 27
6 Disclaimer and version history **32**
6.1 Disclaimer 32
6.2 Version history 32

© 2026 Quorint

Summary

Dev Ojha (Zakura) engaged Quicate to conduct a security assessment of Zakura, a Zcash validator full node implemented in Rust. Between 2 October and 13 October 2026, we reviewed the node end to end across the tagged `v1.6.0` release and `main` branch listed in Section 2 (with delivery-time status rechecked against `v1.7.0-rc0` at `cd103f69`): consensus and block verification, the finalized and non-finalized state, native header-chain and verifiable commitment tree (VCT) fast synchronization, both peer-to-peer stacks (the legacy TCP stack and the v2 QUIC/ `iroh` stack), the JSON-RPC interface, the forked cryptography crates, and the release and deployment automation.

We identified **35 findings** across the workspace: **1 Critical**, **1 High**, **13 Medium**, **12 Low**, and **8 Informational**. Each square below represents one finding in report order.



Critical **1** High **1** Medium **13** Low **12** Informational **8**

The core transaction, script, and proof verifiers reject malformed inputs as expected, and several availability findings reflect fail-closed designs that halt rather than commit unverified state. The highest-severity risks concentrate at the boundary between consensus verification and the surrounding header-chain, networking, and state-commit layers:

- **Block-invalidity recording, consensus rules, and fork choice.** On a v2 or `dual` node (which includes the project-operated Mainnet nodes running `dual` mode—nine regional bootstrap peers and four broadcast-gateway backends in the deployment manifests, with six of nine reported active in `dual` mode during the engagement—as well as the default on Testnet, Regtest, and custom networks), any peer can supply a forged block body that preserves the header's `hashMerkleRoot` via trailing-transaction duplication or ZIP-244 authorizing-data mutation. Zakura rejects the copy before `hashBlockCommitments` is checked in the state and records a permanent `ConsensusBodyInvalid` tombstone keyed only by the honest block's hash, halting the node at the parent height or splitting it onto a sibling chain (`ZAK-01` ; on the legacy-only stack the same payload forces a repeatable sync restart). In addition, `zakura-header-chain` drops Zebra's first-received fork-choice tie-break in favor of raw `tip_hash.0` bytes (`ZAK-05`), `subsidy_is_valid` returns early on zero block subsidy before checking ZIP-271 lockbox disbursements (`ZAK-10`), and Regtest / `disable_pow` header validation lacks a no-retargeting branch while crediting unproved claimed work (`ZAK-11`).
- **Peer-to-peer source accounting, banning, and handshake/discovery limits.** On the default Mainnet legacy stack, per-source connection limits and bans key on exact 128-bit IPv6 addresses without `/64` prefix grouping, allowing a single `/64` allocation to fill and hold all 300 inbound slots while bans never expire (`ZAK-02`). On the v2 QUIC/ `iroh` stack, misbehavior is record-only with no ban table or cross-stack enforcement (`ZAK-07`), a single global 32-permit handshake semaphore is acquired before per-IP checks and shared across inbound and outbound dials (`ZAK-08`), discovery verifies up to 32 Ed25519 signatures per `Peers` batch inline and double-verifies `Hello` self-records under a global mutex (`ZAK-09`), and `zakura-iroh` retains an address-table entry per authenticated remote identity for the lifetime of the endpoint (`ZAK-12`).
- **Sync availability, panic containment, RPC, and release pipelines.** Detached Rayon verifier jobs run without `catch_unwind` or a pool `panic_handler`, so any verifier panic calls `std::process::abort()` and bypasses per-peer panic containment (`ZAK-04`); the non-finalized address index credits outputs before debiting inputs and panics the state write thread on a valid self-spend block whose intermediate balance exceeds `MAX_MONEY` (`ZAK-06`); VCT fast sync parks indefinitely if all connected v2 header peers withhold `tree_aux` (`ZAK-03`); Sybil peers can fill staged sync-target slots with unproved work claims (`ZAK-14`); unauthenticated remote RPC endpoints (as configured in the Testnet deployment workflow) have no per-method cost bound (`ZAK-13`); and the release workflow suppresses `zizmor` 's cache-poisoning check across shared BuildKit caches (`ZAK-15`).

© 2026 Quorint

What we reviewed

2.1 The code

We reviewed Zakura across the tagged `v1.6.0` release (`ebda15e23ed0189787e21b4625c6e4f62d10932f` , released 1 October 2026) and the `1.6.1` `main` branch at `38af6a6a462dce431fef729c0d051a79d774b212` (`38af6a6a` , 8 October 2026, including PRs `#950` and `#1250`) through `454f07671376e86f86677168b4f9fe49998c9b3d` (`454f0767` , 11 October 2026, including PRs `#754` , `#1117` , and `#1362`) of `github.com/zakura-core/zakura` , with a delivery-time status recheck against `v1.7.0-rc0` / `main` at `cd103f6968b2120d6410140d4cd3541f7b5d9cf4` (`cd103f69` , 13 October 2026). Unless noted otherwise, line numbers cite the `38af6a6a` baseline.

COMPONENT	LANGUAGE	ROLE IN THE SYSTEM
`zakura-chain`	Rust	Sync-only core types: blocks, transactions, txids, Merkle roots, difficulty, and amounts
`zakura-consensus`	Rust	Semantic block and transaction verification, subsidy/ZIP-271 checks, and checkpoint verifier
`zakura-state`	Rust	Finalized RocksDB state (`zakura_db`), non-finalized chain state, VCT fast sync, and rollback
`zakura-header-chain`	Rust	Native header engine, `ChainScore` fork choice, body records, and auxiliary admission
`zakura-network`	Rust	Legacy TCP stack and v2 QUIC/ `iroh` stack, peer set, handshakes, and discovery
`zakura-rpc` & `zakura-script`	Rust / C++ FFI	JSON-RPC and gRPC surfaces, block templates, and script verification (`libzcash_script` FFI at `38af6a6a` ; pure-Rust `zcash_script` in `cd103f69` / `#1247`)
`zakurad`	Rust	Node CLI, orchestration, header-sync and block-sync drivers, and mempool
Forked crates & CI	Rust / YAML	`zakura-halo2-proofs` , `zakura-pasta-curves` , `zakura-iroh` , `.github/workflows` , and `deploy/`

Zakura descends from Zebra as a validator-focused full node with a strict downward dependency flow from `zakura-chain` through `zakura-consensus` , `zakura-state` , `zakura-network` , `zakura-rpc` , and `zakura-script` up to `zakurad` . Alongside the inherited legacy TCP stack (default on Mainnet), Zakura adds a v2 QUIC/ `iroh` stack (default on Testnet, Regtest, and custom networks, and run in `dual` mode on most of the project's Mainnet nodes), a native header-chain engine, and verifiable commitment tree (VCT) fast sync.

2.2 What we did not cover

- The end-of-life `zcashd` implementation, and any threat model in which the attacker is a trusted node operator or administrator.
- A first-principles circuit audit of the upstream Halo 2 proving system (we reviewed the forked GLV/Pasta integration; see `ZAK-35`).
- Live exploitation or denial-of-service testing against production peers, miners, or the live fleet; all dynamic verification was performed offline in isolated local harnesses.

2.3 Timeline and people

DATE	WHAT HAPPENED
2 October 2026	Kick-off and start of the primary review (`v1.6.0` / `ebda15e23`)
2 – 11 October 2026	Manual review, reachability tracing, and PoC development across `38af6a6a` and `454f0767`
12 – 13 October 2026	Final adversarial verification pass and delivery-time recheck against `v1.7.0-rc0` (`cd103f69`)
14 October 2026	Delivery of the final assessment report

Ouicate carried out the review for Quorint. Contact: `contact@quorint.xyz`

© 2026 Quorint

How we worked

3.1 What we set out to answer

We evaluated Zakura from the perspective of an untrusted network peer, block producer, or unauthenticated RPC caller and structured the review around four core questions:

1. **State corruption and permanent consensus divergence:** Can an untrusted peer cause a validator or miner node to permanently reject a valid block, accept an invalid block, split from the canonical chain, or corrupt its finalized or non-finalized database?
2. **Fund loss, value inflation, and fork-choice manipulation:** Can any transaction, subsidy/disbursement, or state-commit path lead to fund loss, unauthorized value creation, silent balance-accounting errors, or premature tip displacement?
3. **Peer-to-peer abuse resistance across both stacks:** Do the legacy TCP stack and the v2 QUIC/ `iroh` stack enforce sound per-source connection limits, prefix-grouped bans, and bounded handshake and discovery work?
4. **Panic containment and sync/RPC availability:** Are panics and arithmetic errors on consensus, verifier, and state-commit paths contained, and can a low-cost remote party stall synchronization or exhaust node resources?

3.2 Method

We combined manual line-by-line review of the workspace crates and forked dependencies with end-to-end control- and data-flow tracing from untrusted network, block, and RPC inputs to consensus, state, and storage sinks, backed by targeted unit, property, and differential harnesses on the audited revisions.

AREA	WHAT WE LOOKED FOR
Consensus & state integrity	Agreement across semantic, checkpoint, and header-chain verifiers, finalized/non-finalized commit ordering, VCT frontier safety, `ChainScore` fork choice, and `ConsensusBodyInvalid` tombstones.
Availability & panic containment	Detached Rayon verifier jobs (`spawn_fifo` vs. SR-1 containment), state-commit worker panics, head-of-line stalls in v2 header and block sync, and unauthenticated JSON-RPC cost bounds.
Peer & protocol abuse	Legacy TCP and v2 QUIC/ `iroh` parity: IPv6 `/64` prefix grouping, ban persistence and cross-stack enforcement, handshake permit pools, and Ed25519 discovery verification bounds.
Arithmetic & invariants	Checked vs. `.expect()` arithmetic on consensus and index paths (`Amount` , non-finalized credit-before-debit in `chain.rs` , ZIP-271 lockbox disbursements), and shared-path `assert!` sites.
Supply chain & forked crypto	Native (`zakura-chain`) vs. forked (`zakura-primitives` , `zakura-orchard`) deserializers, custom GLV multi-exponentiation in `zakura-pasta-curves` , and CI build-cache / deployment workflows.

3.3 How we rate

Each finding is rated **Critical**, **High**, **Medium**, **Low**, or **Informational** based on **Impact** (worst-case consequence to consensus integrity, state safety, or node availability) and **Likelihood** (how practical the condition is to trigger under the deployed or default configuration).

RATING	MEANING
■ **Critical**	Can cause a permanent chain split, acceptance of invalid state, permanent consensus halt of deployed nodes, or loss/inflation of funds without privileged access.
□ **High**	Can repeatedly crash or eclipse a default node, stall synchronization across the network, or bypass a primary security boundary at low attacker cost.
■ **Medium**	Degrades node availability, exhausts bounded resources, causes consensus divergence on non-default network configurations, or weakens a defense-in-depth boundary.
□ Low	Has bounded impact, requires non-standard operator action or local access, or affects edge-case protocol compliance.
□ Informational	Code-quality, observability, consistency, or defense-in-depth hardening note with no direct exploit path.

© 2026 Quorint

Findings at a glance

All 35 findings ordered by severity. Status reflects the `38af6a6a` / `454f0767` baseline; post-cutoff updates in `v1.7.0-rc0` (`cd103f69`)— `ZAK-12` resolved (`#1354`), `ZAK-08` mitigated (`#1354`), `ZAK-05` partially remediated (`#1084`), and `ZAK-06` narrowed (`#1366`)—are noted in Section 5.

ID	FINDING	RATING	STATUS
ZAK-01	One forged block body makes a node permanently reject the honest block	■ **Critical**	Open
ZAK-02	One IPv6 `/64` can fill every inbound slot of a node	□ **High**	Open
ZAK-03	VCT fast sync parks indefinitely when every header peer withholds tree data	■ **Medium**	Open
ZAK-04	A panic in a Rayon verifier job aborts the whole node	■ **Medium**	Open
ZAK-05	Fork choice drops Zebra's first-received tie-break and allows 1-block tip displacement	■ **Medium**	Open
ZAK-06	Crediting before debiting lets a valid self-spend block panic the write thread	■ **Medium**	Open
ZAK-07	The v2 stack never bans peers and lets legacy-banned IPs reconnect	■ **Medium**	Open
ZAK-08	One IP can hold all v2 handshake permits	■ **Medium**	Open
ZAK-09	Discovery signature verification has no per-source cost bound	■ **Medium**	Open
ZAK-10	Zero-subsidy early return skips ZIP-271 checks and diverges from the checkpoint path	■ **Medium**	Open
ZAK-11	Regtest difficulty has no no-retargeting branch; the disable-PoW waiver accepts any target	■ **Medium**	Open
ZAK-12	`iroh` keeps an address-table entry per authenticated identity forever	■ **Medium**	Open
ZAK-13	With cookie auth disabled, unauthenticated RPCs have no per-method cost bound	■ **Medium**	Open
ZAK-14	A peer can fill staged sync-target slots with fabricated work claims	■ **Medium**	Open
ZAK-15	Release workflow suppresses the cache-poisoning check	■ **Medium**	Open
ZAK-16	Windows major upgrade abandons the old database	□ Low	Open
ZAK-17	`invalidateblock` drops siblings and can hang the checkpoint path	□ Low	Open
ZAK-18	Rollback uses unbounded memory and can wipe the backup	□ Low	Open
ZAK-19	Orphaned-block transactions are not restored after a reorg	□ Low	Open
ZAK-20	Gossiped blocks at an unexpected height are dropped without scoring	□ Low	Open
ZAK-21	The mempool crawler bypasses the per-peer request cap	□ Low	Open
ZAK-22	Sapling batch-verification fallback amplifies CPU for a crafted batch	□ Low	Open
ZAK-23	v2 streams have no message tables (per-kind accounting)	□ Low	Open
ZAK-24	QUIC receive windows have no aggregate memory budget	□ Low	Open
ZAK-25	Discovery records do not prove address ownership	□ Low	Open
ZAK-26	A retryable verification error has no give-up bound	□ Low	Open

© 2026 Quorint

ID	FINDING	RATING	STATUS
ZAK-27	Inherited legacy P2P serving limits may be loose for Zakura's model	☐ Low	Open
ZAK-28	Fork parser accepts a pre-NU6.3 v6 branch ID	☐ Informational	Open
ZAK-29	Invariant sites panic on shared (non-per-peer) paths	☐ Informational	Open
ZAK-30	Checkpoint commit has no park path and flags VCT state early	☐ Informational	Open
ZAK-31	Some tree getters return empty trees instead of explicit absence	☐ Informational	Open
ZAK-32	Block-template builder does not check transaction versions	☐ Informational	Open
ZAK-33	Redelivered auxiliary record fails a header batch	☐ Informational	Open
ZAK-34	Orchard proof-size rule differs between native and fork parsers before NU6.2	☐ Informational	Open
ZAK-35	Unaudited GLV multi-exponentiation is live on the proof path; the dormant `orbits` feature has no CI tripwire	☐ Informational	Open

© 2026 Quorint

Findings

Findings are ordered by severity from Critical to Informational. Each entry records its rating, likelihood, impact, remediation status, affected target paths, and category, followed by a technical description, impact analysis, and remediation guidance. Low and Informational findings use a compact layout with description and recommendation.

Critical (1)

ZAK-01

CRITICAL

One forged block body makes a node permanently reject the honest block

RATING	LIKELIHOOD	IMPACT	STATUS
Critical	Medium	Critical	Open

TARGET	CATEGORY
zakura-header-chain · graph/header_node.rs:69-74, graph/mod.rs:247; zakurad · commands/start/zakura/block_sync_driver.rs:1564-1572, 1671-1680; zakura-consensus · block.rs:467-714	Consensus, data validation

Description

A block's hash covers its transactions only through the Merkle root of the transaction IDs (`hashMerkleRoot`). Several changes to a block body leave the header hash unchanged and still make the block invalid. Any peer can construct them from public data without mining proof-of-work:

- **Vector A (CVE-2012-2459).** Duplicate the trailing transaction(s) of a suitable transaction list. The Merkle root stays the same while the block fails the duplicate-transaction check.
- **Vector B (ZIP-244 authorizing data).** Change a V5 transaction's authorizing data: a `scriptSig` hash-type byte, a signature or key byte, or a Sapling or Orchard proof or signature. ZIP-244 leaves authorizing data out of the transaction ID, so every txid and the block hash stay the same.
- **Vector B' (`scriptSig` sigop padding).** Prefix each `scriptSig` with dead `OP_CHECKMULTISIG` padding. The block exceeds the 20,000 signature-operation limit (`MAX_BLOCK_SIGOPS`) and no txid moves.

Zakura rejects every one of these copies correctly. The fault is in what it does next. On the v2 block-sync path it records the rejection as a permanent fact about the honest block's hash. The block-sync driver writes a `ConsensusBodyInvalid` record for the requested hash (`crates/zakurad/src/commands/start/zakura/block_sync_driver.rs:1564-1572`). Its evidence is a digest that binds no byte of the delivered body:

crates/zakurad/src/commands/start/zakura/block_sync_driver.rs:1671-1680

```
fn intrinsic_body_invalid_evidence(
    hash: block::Hash,
    rule: &zakura_header_chain::BodyRuleId,
) -> zakura_header_chain::EvidenceId {
    let mut hasher = Sha256::new();
    hasher.update(b"zakura-consensus-body-invalid-v1");
    hasher.update(hash.0);
    hash_bytes(&mut hasher, rule.as_str().as_bytes());
    zakura_header_chain::EvidenceId::from_digest(hasher.finalize().into())
}
```

The record is written before the one check that could tell a bad supplier from a bad block: the ZIP-244 `hashBlockCommitments` comparison (`block_commitment_is_valid_for_chain_history` in

© 2026 Quorint

`crates/zakura-state/src/service/check.rs:264`), which runs later in the state and so never runs for these copies. The semantic verifier works in this order: Merkle root (`crates/zakura-consensus/src/block.rs:467`), transaction and script checks (`:539-648`), signature-operation limit (`:679`), build with `auth_data_root: None` (`:714`), commit. Every failure returns before the commitment check. `crates/zakura-header-chain/src/graph/header_node.rs:69-74` attaches the `ConsensusBodyInvalid` record to the header node keyed by block hash, and `crates/zakura-header-chain/src/graph/mod.rs:247` raises `PermanentBodyInvalidity` for that hash and all descendants, on every commit path and after restart.

The project's own code anticipates this regression. `crates/zakura-chain/src/block/merkle.rs:55-69` carries Bitcoin's CVE-2012-2459 note warning that a node which "marks that block as permanently invalid … will fail to accept further unmodified (and thus potentially valid) versions of the same block," and stating that the vulnerability "does not apply to Zebra, because it does not store invalid data on disk, and because it does not permanently fail blocks." The header-chain `ConsensusBodyInvalid` record is that permanent failure.

Impact

On a v2 or `dual` node, one body message from any connected peer makes the node permanently refuse an honest block across every commit path (non-finalized, checkpoint, across restarts, and through the legacy fallback on `dual`), halting the node at the parent height. If a valid sibling block exists at that height, the node commits it and stays on a minority fork permanently because every later canonical block descends from the tombstoned hash.

The v2 and `dual` stack is the default on Testnet, Regtest, and custom networks, and is used in `dual` mode on the project-operated Mainnet fleet (nine regional bootstrap peers and four broadcast-gateway backends in the repository manifests; six of nine nodes and three of four gateway backends reported in `dual` mode during the engagement). On the default Mainnet legacy-only stack, no tombstone is written, and the same message forces a free, repeatable sync restart.

Recommendation

Fix the fault at the record rather than at any single validation rule:

1. **Bind authorizing data before recording.** When a peer-supplied block fails with an error that reads authorizing data (`ConsensusInvalid`), and the block commits to that data (NU5 and later), check whether the header commits to this body's authorizing data using `block_commitment_is_valid_for_chain_history`. Record `ConsensusBodyInvalid` only on a proven match; on a mismatch, re-fetch from another peer as a supplier-attributed `PayloadMismatch(AuthDataRoot)`. If the parent history is not yet available, return a retryable error and record nothing. Do not simply suppress recording on authorizing-data errors without penalizing the supplier and re-fetching, or peers can serve corrupted bodies indefinitely while a genuinely invalid block is never tombstoned; and do not alter `SIGHASH_SINGLE` to return a digest when `index >= outputs.len()`, which would reintroduce the legacy `0x0100...00` sighash bug (`GHSA-cwfq-rfcr-8hmp` / `GHSA-pvmv-cwg8-v6c8`).
2. **Treat a duplicated tail as a payload mismatch.** Return `PayloadMismatch(TransactionMerkleRoot)` with peer score 100 on both verifiers, as the CVE-2012-2459 note in `merkle.rs` prescribes.
3. **Recover existing records.** Ship a database-format migration that returns records whose rule could derive from unbound data to a retryable state, and add an operator command that clears a single record by hash.

© 2026 Quorint

High (1)

ZAK-02 HIGH

One IPv6 /64 can fill every inbound slot of a node

RATING	LIKELIHOOD	IMPACT	STATUS
☐ High	High	High	Open

TARGET	CATEGORY
zakura-network · protocol/external/addr/canonical.rs:19-27, peer_set/set/recent_by_ip.rs, peer_set/set.rs:786-790, address_book.rs:35-77	Denial of service, peer management

Description

Every per-source control on the legacy stack keys on one exact, canonicalized `IpAddr`. That covers the inbound attempt limiter, the peer-set per-IP cap, the outbound per-IP check, and the ban list. The only normalization folds IPv4-mapped IPv6 addresses into IPv4; nothing groups IPv6 addresses by routing prefix:

crates/zakura-network/src/protocol/external/addr/canonical.rs:19-27

```
pub(in super::super) fn canonical_ip_addr(v6_addr: &Ipv6Addr) → IpAddr {
    use IpAddr::*;

    // if it is an IPv4-mapped address, convert to V4, otherwise leave it as V6
    v6_addr
        .to_ipv4_mapped()
        .map(V4)
        .unwrap_or_else(|| V6(*v6_addr))
}
```

A host with one routed IPv6 `/64` (the standard allocation for one VPS or broadband line) therefore looks like 2^{64} unrelated sources to the attempt limiter (`recent_by_ip.rs` , default `max_connections_per_ip = 1` at `constants.rs:86` , `MIN_PEER_RECONNECTION_DELAY = 119s` at `constants.rs:153`), the peer-set per-IP cap (`set.rs:786-790`), and the ban list (`address_book.rs:44-77`). The one limit the host shares with everyone else is the global inbound cap:
`peerset_initial_target_size * INBOUND_PEER_LIMIT_MULTIPLIER = 100 * 3 = 300` connections at the defaults (`constants.rs:58, 91` , `config.rs:393-395`). That cap is the resource the attacker exhausts.

Bans make it worse. `BanList` (`address_book.rs:35-64`) is an in-memory `HashSet<IpAddr>` FIFO-capped at `MAX_BANNED_IPS = 20_000` (`constants.rs:414`) with no timestamps. Bans never expire, are lost on restart, cannot be listed or cleared over RPC, and evict the oldest entries first once 20,000 addresses are stored.

Impact

An attacker with one `/64` can take 100% of a default Mainnet node's 300 inbound slots and hold them indefinitely by answering the 59-second heartbeat ping (about 5 pong messages/s total), at negligible cost. The node's ban mechanism can neither evict nor deter the attacker: every new address from the `/64` resets the attempt counter and escapes the ban set. In addition, one malformed message from a shared CGNAT, VPN, or Tor-exit IPv4 address bans every honest peer behind that address until restart (or until 20,000 later bans evict it). Because the victim keeps its outbound peers and stays synced, we rate this High rather than Critical.

Recommendation

1. Key every per-source control on a configurable network-prefix group, defaulting to `/64` for IPv6 and `/32` for IPv4, so one allocation counts as one source for the attempt limiter, the per-IP cap, and the ban set.
2. Give bans a timestamped expiry and a bounded, persisted store, and expose list and clear methods over the admin RPC.

© 2026 Quorint

3. Account inbound slots per prefix group so no single allocation can take the whole inbound budget.

Medium (13)

ZAK-03 MEDIUM

VCT fast sync parks indefinitely when every header peer withholds tree data

RATING	LIKELIHOOD	IMPACT	STATUS
■ **Medium**	Low	Medium	Open

TARGET	CATEGORY
`zakura-state · service/finalized_state.rs:449-466, 1199-1214, service/write.rs:2167-2191, 2443-2469, service/write/vct_write_retry.rs:233-274`	Availability, VCT fast sync

Description

Below the last checkpoint, VCT fast sync does not rebuild the note-commitment trees locally. Instead, it folds in the Sapling, Orchard, and Ironwood roots that peers deliver as `tree_aux`, with each root authenticated by the successor header's `hashBlockCommitments`.

After the first fast block commits, the stored note-commitment frontier is frozen (`self.vct.start_vct_sync_below_last_checkpoint()`). Recomputing a later height locally against that stale frontier would fold a wrong root into the history MMR and corrupt state. The committer therefore refuses to recompute when a height has no verifiable supplied root, returning `VctSuppliedRootUnavailable`:

crates/zakura-state/src/service/finalized_state.rs:1199-1214

```
} else if self.vct.is_below_last_checkpoint() {
    // Frozen-frontier safety: fast sync already froze the note-commitment
    // frontier. This height has no valid supplied root because no peer supplied
    // one or verification evicted it. Local recomputation would fold an
    // incorrect root into the history MMR and corrupt state.
    metrics::counter!("state.vct.root.unavailable.count").increment(1);
    return Err(
        ValidateContextError::VctSuppliedRootUnavailable { height }.into()
    );
}
```

The error is classified as retryable (`write.rs:2167-2191`). The writer restores the parent trees and parks the block, waiting for a header-chain insertion with no terminal deadline after a 30-second diagnostic warning (`write/vct_write_retry.rs:233-274`).

Impact

The design correctly fails closed, so nothing invalid is written; the cost is availability. On a Mainnet node with the v2 stack opted in (including the `dual` deployment configuration), if every connected v2 header peer withholds or corrupts `tree_aux` for one height below the checkpoint, finalized commits stop with no deadline. A restart stays frozen, and disabling `vct_fast_sync` after the frontier is frozen causes the node to refuse to start (`finalized_state.rs:449-466`). The node recovers only when a v2 peer that serves valid `tree_aux` connects. A connection-budget eclipse built from `ZAK-02`, `ZAK-07`, and `ZAK-08` makes the withholding precondition cheap and durable.

Recommendation

1. Bound the VCT park wait with an escalating deadline that raises an operator alarm and, where safe, rebuilds the note-commitment trees from a trusted checkpoint or genesis frontier.
2. Treat persistent `tree_aux` withholding or invalid repair responses across connected suppliers as a scored peer fault so the node rotates header peers.
3. Provide a supported recovery path when `vct_fast_sync` is disabled on a database with a frozen frontier instead of refusing to start without manual intervention.

© 2026 Quorint

ZAK-04

MEDIUM

A panic in a Rayon verifier job aborts the whole node

RATING	LIKELIHOOD	IMPACT	STATUS
■ Medium	Low	High	Open

TARGET	CATEGORY
zakurad · application.rs:427-436; zakura-consensus · primitives.rs:223-234; zakura-network · zakura/transport/pipe.rs:289-296, 330-335	Availability, panic containment

Description

Most signature, Halo 2 / Groth16 proof, and transparent-script checks run as detached Rayon jobs via `zakura_consensus::primitives::spawn_fifo` (`crates/zakura-consensus/src/primitives.rs:223-234`). The global Rayon thread pool is built without a custom `panic_handler` (`crates/zakurad/src/application.rs:427-436`), and `spawn_fifo` does not wrap its closure in `std::panic::catch_unwind`.

In `rayon-core 1.13.0`, an unhandled panic in a detached `spawn_fifo` job with no pool `panic_handler` calls `std::process::abort()`. That is an immediate process kill that runs no destructors and bypasses the workspace's `panic = "unwind"` setting. Meanwhile, the v2 per-peer pipe layer explicitly codifies SR-1 ("blast radius is exactly one peer") with a compile-time guard (`crates/zakura-network/src/zakura/transport/pipe.rs:289-296, 330-335`), which the unguarded Rayon pool defeats from the inside:

`crates/zakurad/src/application.rs:427-436 · crates/zakura-network/src/zakura/transport/pipe.rs:330-335`

```
// application.rs:
rayon::ThreadPoolBuilder::new()
    .num_threads(config.sync.parallel_cpu_threads)
    .thread_name(|thread_index| format!("rayon {thread_index}"))
    .build_global()
    .expect("unable to initialize rayon thread pool"); // no .panic_handler(...)

// pipe.rs:330-335 (SR-1 — "blast radius is exactly one peer"):
#[cfg(panic = "abort")]
compile_error!(
    "Zakura peer-pipe panic containment requires `panic = \"unwind\"` \
     (security_requirements.md SR-1); this build sets `panic = \"abort\"`. \
     Set panic = \"unwind\" in the workspace [profile.dev] and [profile.release]."
);
```

Impact

Any verifier-job panic aborts the entire `zakurad` process on every network and either P2P stack. No attacker-reachable verifier panic was found in the current code (51 million adversarial inputs plus the project's real-proof Halo 2 harness closed every tested candidate), though an environmental abort is reachable on hardened or containerized deployments if OS-RNG initialization fails. If a verifier panic ever becomes reachable through a future fork change, dependency update, or internal assertion, a single gossiped block or transaction would abort every validating node.

Recommendation

1. Wrap the closure inside `zakura_consensus::primitives::spawn_fifo` in `std::panic::catch_unwind(AssertUnwindSafe(f))` and install a Rayon `panic_handler` in `application.rs` so any verifier panic maps to a contained verification error consistent with SR-1.
2. Ensure the same `catch_unwind` containment covers the script verifier boundary (`zakura-script`, which migrated from the C++ `libzcash_script` FFI to pure-Rust `zcash_script` in `cd103f69` / `#1247`).
3. Add a regression test verifying that a panicking `spawn_fifo` job returns `Err` on its oneshot channel without aborting the test runner.

© 2026 Quorint

ZAK-05

MEDIUM

Fork choice drops Zebra's first-received tie-break and allows 1-block tip displacement

RATING	LIKELIHOOD	IMPACT	STATUS
■ Medium	High	Medium	Open

TARGET	CATEGORY
zakura-header-chain · graph/frontier.rs:91-98, graph/mod.rs:822-845; zakura-state · service/non_finalized_state/chain.rs:2553-2616	Consensus, fork choice

Description

Zakura's entire fork-choice ordering in `zakura-header-chain` (`frontier.rs:91-98`) and `zakura-state` (`chain.rs:2553-2616`) is `(suffix_work, raw_tip_hash_bytes)`. Upstream Zebra (`ZcashFoundation/zebra#11341`) and the Zcash protocol specification insert a first-received layer (`received_time`) before the tip-hash tie-break so an already-adopted tip is not displaced by an equal-work sibling that arrives later.

Sibling blocks at the same height on Zcash always carry identical work, because `nBits` (`difficulty_threshold`) is a deterministic function of the shared parent window. The tie-break therefore fires on every natural 1-block fork and selects a tip purely by comparing the raw 32-byte internal `tip_hash.0` array:

crates/zakura-header-chain/src/graph/frontier.rs:91-98

```
impl Ord for ChainScore {
    fn cmp(&self, other: &Self) → Ordering {
        self.suffix_work
            .cmp(&other.suffix_work)
            .then_with(|| self.tip_hash.0.cmp(&other.tip_hash.0))
    }
}
```

Impact

In roughly 50% of 1-block fork races—those where arrival order and raw-hash order disagree—a Zakura node transiently selects a different best tip than the Zebra-dominant network. Moreover, an equal-work sibling that arrives arbitrarily later will deterministically displace an already-adopted tip whenever its raw `tip_hash.0` bytes compare greater, creating a transient 1-block reorganization (or a 1-confirmation double-spend window against a zero- or one-confirmation acceptor if an attacker mines a sibling with a larger raw hash). Although `454f0767` (`#1117`) adds an operator-invoked `preciousblock` RPC override in `zakura-state`, automatic fork choice for unmarked equal-work tips at `454f0767` in both `zakura-header-chain` and `zakura-state` remains keyed on `tip_hash.0` (subsequently, `cd103f69` / `#1084` restores `receipt_order` first-received tie-breaking in `zakura-state`, while `zakura-header-chain` `ChainScore::cmp` remains keyed on `tip_hash.0`).

Recommendation

Restore Zebra's first-received preference in `ChainScore` (preferring the earliest-received equal-work tip and falling through to `tip_hash` only when two branches are evaluated without a distinct receipt order) so an already-adopted tip cannot be displaced by a later equal-work sibling.

© 2026 Quorint

ZAK-06

MEDIUM

Crediting before debiting lets a valid self-spend block panic the write thread

RATING	LIKELIHOOD	IMPACT	STATUS
■ Medium	Low	High	Open

TARGET	CATEGORY
zakura-state · service/non_finalized_state/chain.rs:1840-1843, service/non_finalized_state/chain/index.rs:72-79, service/write.rs:1948-1982	Arithmetic, state commit

Description

When a block commits to the non-finalized state, `Chain::update_chain_tip_with_block_except_trees` updates the per-address transparent balance index by crediting each transaction's newly created outputs **before** debiting its spent inputs:

crates/zakura-state/src/service/non_finalized_state/chain.rs:1840-1843

```
// add the utxos this produced
self.update_chain_tip_with(&(outputs, &transaction_hash, new_outputs))?;
// delete the utxos this consumed
self.update_chain_tip_with(&(inputs, &transaction_hash, spent_outputs))?;
```

The running per-address balance is stored as an `Amount<NegativeAllowed>` constrained to `[-MAX_MONEY, +MAX_MONEY]`. The credit step adds each output's value to `self.balance` and calls `.expect("total UTXO value has already been checked")` on the checked addition (`crates/zakura-state/src/service/non_finalized_state/chain/index.rs:72-79`).

In a same-address self-spend, crediting the new output before debiting the spent input produces an intermediate balance of `balance_before + output_value`. Whenever that intermediate sum exceeds `MAX_MONEY` (2.1×10^{15} zatoshis), `Amount::add` returns `Err(ConstraintError)` and `.expect(...)` panics. Notably, the finalized RocksDB twin in `crates/zakura-state/src/service/finalized_state/zakura_db/transparent.rs:440-451, 520-550` already debits inputs before crediting outputs and documents this exact overflow hazard.

Impact

The triggering block is 100% consensus-valid: value is conserved and no consensus rule bounds a single address's holdings below `MAX_MONEY`. The block passes semantic verification, and the panic fires during state commit on the block-write worker thread (`crates/zakura-state/src/service/write.rs:1948-1982`). That thread calls `resume_unwind` and latches `BlockWriteTaskFailure::panic()`, halting all further finalized and non-finalized commits; restarting the node re-downloads the canonical block and panics at the same height. On Regtest, custom, and test networks the trigger is cheap; on Mainnet it requires a single transparent address holding `> MAX_MONEY / 2` (10.5M ZEC) (and after `cd103f69` / `#1366`, non-finalized transparent address indexing runs only when address indexing or `lightwalletd` is enabled or on unpruned state).

Recommendation

Mirror the finalized-state ordering in `crates/zakura-state/src/service/non_finalized_state/chain.rs:1840-1843`: debit a transaction's spent inputs before crediting its created outputs.

© 2026 Quorint

ZAK-07

MEDIUM

The v2 stack never bans peers and lets legacy-banned IPs reconnect

RATING	LIKELIHOOD	IMPACT	STATUS
■ Medium	High	Medium	Open

TARGET	CATEGORY
zakura-network · zakura/block_sync/peer_routine.rs:2290-2303, zakura/handler.rs:1326-1338, 2224-2286; zakurad · commands/start/zakura/block_sync_driver.rs:442-445	Peer management, DoS

Description

The v2 QUIC/ `iroh` stack has no banning path and no cross-stack exclusion mechanism. In block sync, misbehavior is explicitly record-only:

crates/zakura-network/src/zakura/block_sync/peer_routine.rs:2290-2293 · block_sync_driver.rs:442-445

```
// peer_routine.rs:
async fn report_misbehavior(&self, reason: BlockSyncMisbehavior) {
    // Misbehavior is record-only: observe and forward it, but never cancel the
    // session. Peer scoring no longer drives disconnects.
    metrics::counter!("sync.block.peer.violation").increment(1);

// block_sync_driver.rs:
BlockSyncAction::Misbehavior { peer, reason } => {
    // Record-only: peer scoring no longer drives disconnects.
    debug!(?peer, ?reason, "recorded Zakura block-sync peer violation");
}
```

In header sync, protocol violations are also traced as record-only (`header_sync/reactor.rs:4286-4289, 5231-5276`), but the v2 admission handler (`handler.rs:1326-1338, 2224-2286`) keeps no ban table and does not consult the legacy TCP `BanList` . A peer that disconnects or is banned on the legacy stack can immediately reconnect over v2 with the same IP and key.

Impact

Because misbehaving v2 peers are never excluded across sessions, every per-connection abuse vector on the v2 stack can be repeated indefinitely at zero penalty. This directly multiplies `ZAK-01` (forged block bodies), `ZAK-08` (handshake-permit exhaustion), `ZAK-09` (discovery signature floods), and `ZAK-14` (staged target pinning), and allows legacy-banned hosts to continue interacting with `dual` nodes over port 8234.

Recommendation

Give the v2 stack a scored ban and cooldown exclusion table shared with the legacy stack, keyed on `/32` IPv4 and `/64` IPv6 prefixes as well as v2 `ZakuraPeerId` , and reject banned sources during v2 admission.

© 2026 Quorint

ZAK-08

MEDIUM

One IP can hold all v2 handshake permits

RATING	LIKELIHOOD	IMPACT	STATUS
■ **Medium**	Medium	Medium	Open

TARGET	CATEGORY
zakura-network · zakura/handler.rs:81-83, 122, 1326-1338, 2294-2306, 4008-4013	Denial of service, v2 handshake

Description

The v2 handler bounds concurrent control handshakes across both inbound and outbound connections with a single global semaphore of 32 permits (`DEFAULT_ZAKURA_MAX_PENDING_HANDSHAKES = 32` , `crates/zakura-network/src/zakura/handler.rs:83`). Inbound connections acquire one of those 32 permits in `accept_native_responder_handshake` (`handler.rs:2294-2306`) **before** `register_session` checks the per-IP active-connection cap (`DEFAULT_ZAKURA_MAX_CONNS_PER_IP = 16` , `handler.rs:81, 1326-1338`):

`crates/zakura-network/src/zakura/handler.rs:2294-2306`

```
let Ok(_handshake) = self.pending_handshakes.clone().try_acquire_owned() else {
    metrics::counter!("zakura.p2p.conn.rejected.pending_handshake").increment(1);
    conn.trace_connection("rejected.admission", None, Some("inbound"),
Some("pending_handshake"));
    connection.close(VarInt::from_u32(ZAKURA_CLOSE_RESOURCE), b"pending handshake");
    return Err(ZakuraHandlerError::ResourceLimit("pending handshake"));
};
```

The responder handshake then runs through up to five sequential `control_timeout` windows of 10 seconds each (`DEFAULT_ZAKURA_CONTROL_TIMEOUT = 10s` at `handler.rs:122` ; `accept_bi` plus `read_control_payload` / `write_control_payload` length-and-payload steps at `handler.rs:2326-2358, 4751-4777`), allowing a slow peer to hold each permit for 30 to 50 seconds. Because outbound dials acquire from the same `self.pending_handshakes` semaphore (`handler.rs:4008-4013`), holding all 32 permits blocks both inbound and outbound v2 handshakes.

Impact

A single IP can hold all 32 v2 handshake permits continuously by opening about 0.64 to 1.07 stalled QUIC connections per second, starving all new inbound and outbound v2 handshakes. On a `dual` node the legacy TCP stack continues to make progress; on a Zakura-only node (`p2p_stack = "zakura"`), new peer acquisition halts completely. (Note: `cd103f69` / `#1354` adds pre-TLS per-IP connection and pending-handshake checks plus stateless QUIC `Retry` in `zakura-quic` , though the 32-permit control-handshake pool remains shared across inbound and outbound dials and per-IP caps still lack IPv6 `/64` grouping.)

Recommendation

Enforce a per-IP / per- `/64` cap on pending handshakes before acquiring from `pending_handshakes` , and separate the outbound dial handshake budget from the untrusted inbound handshake pool.

© 2026 Quorint

ZAK-09

MEDIUM

Discovery signature verification has no per-source cost bound

RATING	LIKELIHOOD	IMPACT	STATUS
■ **Medium**	Medium	Medium	Open

TARGET	CATEGORY
`zakura-network · zakura/discovery/protocol.rs:1821-1852, 1957-1974, 3169-3171`	Denial of service, v2 discovery

Description

The v2 discovery service verifies Ed25519 signatures on `ZakuraNodeRecord` payloads along two paths:

1. When a `Peers` response arrives, `import_peer_records` (`crates/zakura-network/src/zakura/discovery/protocol.rs:1821-1852`) verifies up to `max_imported_records_per_response = 32` Ed25519 signatures inline on the async worker task outside the mutex, bounded only by the generic stream rate limit of 2,048 messages/s (`DEFAULT_ZAKURA_MESSAGE_RATE_PER_SECOND` at `handler.rs:95`).
2. When a peer presents a self-record in `Hello`, `import_connected_peer_record` (`protocol.rs:1957-1974`) acquires `self.inner.lock().await` at line 1959, calls `record.verify(&context)?` at line 1961 while holding the mutex, and then calls `inner.book.import_record(...)` at line 1973 with `pre_verified = false`, which runs `record.verify(context)?` a second time under the same mutex (`protocol.rs:3169-3171`).

Impact

Connected v2 peers can drive sustained Ed25519 verification CPU load via unsolicited or repeated `Peers` batches and contend the global discovery mutex via repeated `Hello` handshakes. The effect is transient and self-healing once traffic stops.

Recommendation

Verify the `Hello` record once **before** acquiring `self.inner.lock().await` and pass it into `import_record_inner` with `pre_verified = true`. Bound `Peers` record verification per source prefix and reject unsolicited `Peers` messages that do not match an outstanding `GetPeers` request.

ZAK-10

MEDIUM

Zero-subsidy early return skips ZIP-271 checks and diverges from the checkpoint path

RATING	LIKELIHOOD	IMPACT	STATUS
■ **Medium**	Medium	Medium	Open

TARGET	CATEGORY
`zakura-consensus · block/check.rs:178-185, 276-297, 358-383, checkpoint.rs:131-134`	Consensus, subsidy & disbursements

Description

ZIP-271 requires the NU6.1 activation block to pay configured one-time lockbox disbursement outputs, reducing the deferred funding-stream pool by their total D. In the semantic block verifier, `subsidy_is_valid` returns early with a zero deferred-pool change whenever `expected_block_subsidy.is_zero()`, before reaching the ZIP-271 disbursement check at lines 276-297:

© 2026 Quorint

crates/zakura-consensus/src/block/check.rs:178-185

```
pub fn subsidy_is_valid(
    block: &Block,
    net: &Network,
    expected_block_subsidy: Amount<NonNegative>,
) -> Result<DeferredPoolBalanceChange, BlockError> {
    if expected_block_subsidy.is_zero() {
        return Ok(DeferredPoolBalanceChange::zero());
    }
    // ...
```

By contrast, the checkpoint verifier unconditionally subtracts `lockbox_disbursement_total_amount(height)` from the deferred pool change:

crates/zakura-consensus/src/checkpoint.rs:131-134

```
Ok(expected_deferred_amount
    .unwrap_or_default()
    .checked_sub(network.lockbox_disbursement_total_amount(height))
    .map(DeferredPoolBalanceChange::new))
```

Impact

The semantic and checkpoint verifiers disagree on any configured custom or Regtest network whose NU6.1 activation height falls in a zero-subsidy halving era (halving ≥ 30) while lockbox disbursements are configured. Because `subsidy_is_valid` returns `DeferredPoolBalanceChange::zero()` early, `miner_fees_are_valid` (`check.rs:358-383`) checks `total_output_value == total_input_value` with zero subsidy and zero deferred-pool deduction: a node verifying semantically accepts an invalid block whose coinbase omits the mandatory ZIP-271 disbursements and rejects a valid block whose coinbase pays them (including Zakura's own `getblocktemplate` output), whereas a checkpoint-synced node subtracts D from the deferred pool. Mainnet and default Testnet are unaffected because their NU6.1 activation heights occur in non-zero-subsidy eras.

Recommendation

Enforce the ZIP-271 one-time lockbox disbursement check (`check.rs:276-297`) before the `expected_block_subsidy.is_zero()` early return (or return `checked_sub(net.lockbox_disbursement_total_amount(height))` after verifying the disbursement outputs) so the semantic and checkpoint paths agree at every height.

ZAK-11

MEDIUM

Regtest difficulty has no no-retargeting branch; the disable-PoW waiver accepts any target

RATING	LIKELIHOOD	IMPACT	STATUS
■ **Medium**	Medium	Medium	Open

TARGET	CATEGORY
zakura-header-chain · validation/contextual/adjusted_difficulty.rs:200-251, validation/contextual/validate.rs:76-83, validation/prepare/pipeline.rs:173-177, validation/context_free/pow_policy.rs:52-57; zakura-state · service/read/difficulty.rs:274-280	Consensus, difficulty validation

Description

Two difficulty-validation defects exist in `zakura-header-chain` for Regtest and custom `disable_pow` networks:

1. `AdjustedDifficulty::expected_difficulty_threshold` in `crates/zakura-header-chain/src/validation/contextual/adjusted_difficulty.rs:200-251` (which

© 2026 Quorint

is also called by `crates/zakura-state/src/service/read/difficulty.rs:274-280` for `getblocktemplate`) has no `Network::Regtest` no-retargeting branch and runs the full Digishield v3 calculation past the 17-block averaging window, unlike upstream `zcashd` (`fPowNoRetargeting = true`) and `Zebra`, which pin Regtest to `target_difficulty_limit()` (`0x200f_ffff`).

2. When `network.disable_pow()` is true, `crates/zakura-header-chain/src/validation/contextual/validate.rs:76-83` waives the `expected_difficulty` equality check and verifies only that `difficulty_threshold.to_work().is_some()`. Then `crates/zakura-header-chain/src/validation/prepare/pipeline.rs:173-177` credits the full `to_work()` of the header's claimed `difficulty_threshold` into `ChainScore`:

`crates/zakura-header-chain/src/validation/contextual/validate.rs:76-83 · pipeline.rs:173-177`

```
// validate.rs:
if network.disable_pow() {
    if difficulty_threshold.to_work().is_none() {
        return Err(ContextualValidationError::InvalidDifficultyThreshold {
            difficulty_threshold,
            expected_difficulty: network.target_difficulty_limit().to_compact(),
        });
    }
}
// pipeline.rs:
let block_work = header
    .difficulty_threshold
    .to_work()
    .ok_or_else(|| invalid(offset, HeaderRule::Work, "invalid compact target"))?;
```

The waiver correctly refuses to apply on Mainnet and the default Testnet (`validation/context_free/pow_policy.rs:52-57`).

Impact

On Regtest or a custom network with `disable_pow` enabled, any peer or miner can stamp an arbitrary compact target (such as maximum difficulty) into a header without solving Equihash, and `zakura-header-chain` credits that unproved claimed work into `ChainScore` and fork choice. Conversely, if `disable_pow` is false on a Regtest configuration, Zakura retargets `nBits` via Digishield after the 17-block averaging window (height ≥ 18) while `zcashd` and `Zebra` keep the fixed `0x200f_ffff` limit, causing cross-implementation Regtest harnesses to reject each other's blocks. Neither path is reachable on Mainnet or default Testnet.

Recommendation

Add an explicit Regtest / no-retargeting branch in `AdjustedDifficulty::expected_difficulty_threshold` returning `network.target_difficulty_limit().to_compact()`, and when `disable_pow` is active, either enforce `expected_difficulty` or credit a fixed minimal work constant rather than the unproved `difficulty_threshold.to_work()`.

ZAK-12

MEDIUM

iroh keeps an address-table entry per authenticated identity forever

RATING	LIKELIHOOD	IMPACT	STATUS
■ **Medium**	Medium	Medium	Open

TARGET	CATEGORY
`zakura-iroh 1.1.0-rc.1 · src/socket/mapped_addrs.rs:335-353, src/socket/remote_map.rs:45-48; zakura-network · zakura/handler.rs:3494-3496, 3878-3880`	Resource exhaustion, v2 transport

Description

The pinned `zakura-iroh = "=1.1.0-rc.1"` crate (`Cargo.toml:97`) maps each remote `EndpointId` to a synthetic address in `MappedAddrs::endpoint_addrs`. Its `AddrMap::get` method inserts into both `addrs`

© 2026 Quorint

and `lookup` on a miss and exposes only `get` and `lookup` —with no removal or eviction API—while `MAX_INACTIVE_NODES` is commented out (`remote_map.rs:45-48`):

`zakura-iroh 1.1.0-rc.1 · src/socket/mapped_addrs.rs:335-353`

```
pub(super) fn get(&self, key: &K) -> V {
    let mut inner = self.inner.lock().expect("poisoned");
    match inner.addrs.get(key) {
        Some(addr) => *addr,
        None => {
            // ... generate an unused `addr` ...
            inner.addrs.insert(key.clone(), addr);
            inner.lookup.insert(addr, key.clone());
            addr
        }
    }
}
```

Because `ZakuraProtocolHandler` implements only `ProtocolHandler::accept` (`crates/zakura-network/src/zakura/handler.rs:3494-3496`) and the `Router` has no incoming filter (`handler.rs:3878-3880`), `iroh` completes the QUIC TLS 1.3 handshake and inserts the remote identity into `endpoint_addrs` **before** Zakura's admission, pending-handshake, or per-IP checks run. The project documents this in `qa/supply-chain/config.toml:432-435` (about 100–200 bytes retained per distinct authenticated identity for the life of the endpoint).

Impact

On any v2-enabled node at `38af6a6a` / `454f0767`, a single IP can generate free Ed25519 keys and complete QUIC handshakes to grow `endpoint_addrs` monotonically until restart while keeping per-remote actors alive for their 60-second idle timeout. (Resolved post-cutoff in `cd103f69` / `#1354`, which replaces `zakura-iroh` with `crates/zakura-quic`.)

Recommendation

1. Adopt an upstream `iroh` fix that re-enables `MAX_INACTIVE_NODES` pruning on `endpoint_addrs`.
2. Install a per-IP / per-`/64` rate-limiting incoming filter on the `iroh` `Router` (using QUIC `Retry`) so fresh-identity handshakes are throttled before `endpoint_addrs` insertion.

ZAK-13

MEDIUM

With cookie auth disabled, unauthenticated RPCs have no per-method cost bound

RATING	LIKELIHOOD	IMPACT	STATUS
■ **Medium**	Medium	Medium	Open

TARGET	CATEGORY
`zakura-rpc · server.rs:76-83, 167, methods/types/get_block_template.rs:84, 1032-1081; zakura-state · service/read/difficulty.rs:108-111; .github/workflows/zakura-testnet-deploy.yml:174-175, deploy/deployer/deploy.py:53, 69, 756-758`	Denial of service, RPC

Description

With cookie authentication disabled, unauthenticated RPC clients can invoke heavy JSON-RPC methods with no per-method cost or concurrency bound. While normal `getblocktemplate` template construction is serialized to one flight (`MAX_TEMPLATE_BUILDS = 1` at `methods/types/get_block_template.rs:84`), `getblocktemplate` in `proposal` mode (`validate_block_proposal`, `methods/types/get_block_template.rs:1032-1081`) bypasses `MAX_TEMPLATE_BUILDS`, skips PoW and Equihash checks (`block.rs:416-424`), and dispatches arbitrary 2 MiB block bodies directly onto the 5-slot `block_verifier_router` shared with P2P block sync. Likewise, `getnetworksolps` / `getnetworkhashps` accept an unclamped `num_blocks: i32` and walk ancestor headers back to genesis on the Tokio blocking pool (`difficulty.rs:108-111`), alongside unthrottled `getblock` (`verbosity = 2`), `getaddresstxids`, `getaddressutxos`, and `z_getsubtreesbyindex`. Although `enable_cookie_auth` defaults to `true` and

© 2026 Quorint

`listen_addr` defaults to loopback in code, the project's Testnet deployment workflow (`.github/workflows/zakura-testnet-deploy.yml:174-175`, rendered via `deploy/deployer/deploy.py:53, 69, 756-758`) writes `rpc_listen_addr = "0.0.0.0:18232"` and `rpc_enable_cookie_auth = false` (whereas `.github/workflows/zakura-mainnet-deploy.yml:396-405` sets `manage_config = false`).

Impact

Where the RPC port is reachable by untrusted clients with cookie authentication disabled, a remote caller can issue concurrent `getblocktemplate` (`proposal`), `getnetworksolps`, or heavy read calls to contend with P2P block verification and saturate CPU, memory, and RocksDB read threads. On a default loopback listener the impact is Low.

Recommendation

Enforce per-method concurrency and range/cost bounds independent of authentication (gate `validate_block_proposal` behind a dedicated permit, clamp `num_blocks` in `getnetworksolps` / `getnetworkhashps`, and bound address/subtree range queries), and refuse non-loopback `rpc_listen_addr` binds unless authentication is enabled.

ZAK-14

MEDIUM

A peer can fill staged sync-target slots with fabricated work claims

RATING	LIKELIHOOD	IMPACT	STATUS
■ **Medium**	Medium	Medium	Open

TARGET	CATEGORY
`zakura-network` · `zakura/header_sync/scheduler/peer_work.rs:545-563`, `zakura/header_sync/reactor.rs:1704-1724, 5102-5119`	Availability, header sync

Description

In v2 header sync, `PeerWorkScheduler` stages peer-advertised chain tips in a table bounded by `MAX_STAGED_TARGETS_V1` (`crates/zakura-header-chain/src/config.rs:42`). When the table is full, `queue_work` (`crates/zakura-network/src/zakura/header_sync/scheduler/peer_work.rs:545-563`) evicts the lowest-priority `AwaitingLocator` entry based on the unverified `claimed_work` advertised by the peer in `Status` or `TipUpdate`:

`crates/zakura-network/src/zakura/header_sync/scheduler/peer_work.rs:545-563`

```
if self.work_by_peer.len() >= MAX_STAGED_TARGETS_V1 {
    let replace = self
        .work_by_peer
        .iter()
        .filter_map(|(peer, work)| match work {
            PeerWorkState::AwaitingLocator {
                priority: current, ..
            } if *current < priority => Some((peer.clone(), *current)),
            _ => None,
        })
        // ...
}
```

Because `priority` ranks unproven work claims and `retire_peer_work` (`reactor.rs:1704-1724, 5102-5119`) does not penalize or ban a peer whose claimed target fails to materialize (`ZAK-07`)—and replying `Busy` (`0×05`) before the 30-second request deadline (`reactor.rs:1704-1724`) retires the work with zero timeout strikes, bypassing the 3-strike / 60-second timeout cooldown (`reactor.rs:3807-3817`)—Sybil peers can advertise maximal `claimed_work` values to evict honest `AwaitingLocator` entries and occupy all `Active` and `AwaitingLocator` staged target slots.

© 2026 Quorint

Impact

On the default Mainnet (`legacy`) and `dual` configurations, the legacy sync fallback continues to advance the chain. On an opt-in Zakura-only node (`p2p_stack = "zakura"`), filling the staged sync-target slots with fabricated high-work claims stalls header synchronization indefinitely.

Recommendation

1. Bound staged target slots per source prefix and reserve slots for previously verified suppliers.
2. Penalize and evict peers whose advertised `claimed_work` fails to materialize after a locator request.
3. Give the Zakura-only stack a recovery rotation so unproven high-work claims cannot monopolize the scheduler.

ZAK-15

MEDIUM

Release workflow suppresses the cache-poisoning check

RATING	LIKELIHOOD	IMPACT	STATUS
■ **Medium**	Low	High	Open

TARGET	CATEGORY
`.github/workflows/release-binaries.yml:70-78, 134, 150-156, 297, 309-323, 588, 731; .github/workflows/zakura-mainnet-deploy.yml:26-32, 242-273, 321-328; docker/Dockerfile:168-169; deploy/deployer/deploy.py:32, 413-427`	Supply chain, CI / release

Description

In `.github/workflows/release-binaries.yml` , `docker/Dockerfile:168-169` (`171-172` in `v1.6.0`) mounts unkeyed BuildKit caches (`--mount=type=cache,target=${CARGO_TARGET_DIR}` and `--mount=type=cache,target=${CARGO_HOME}`) inside the shared Depot project `gjcfbjk8wc` (`release-binaries.yml:134, 150-156, 309-323`), which is also used by `workflow_dispatch` `dry_run` builds of arbitrary `checkout_ref` commits (`release-binaries.yml:70-78`), while the non-Depot `setup-buildx-action` fallback steps suppress `zizmor` 's `cache-poisoning` check (`# zizmor: ignore[cache-poisoning]` at `release-binaries.yml:297, 588, 731`). In addition, `.github/workflows/zakura-mainnet-deploy.yml:26-32, 242-273, 321-328` builds and deploys arbitrary `ref` commits on the persistent self-hosted runner `zakura-mainnet-deployer` (`us-east-0`) without tag or signature verification and caches binaries at `/root/.cache/zakura-deployer-binaries/zakurad-<sha>` (`.build-cache/zakurad-<sha>` by default in `deploy/deployer/deploy.py:32, 413-427`) by filename alone.

Impact

Exploitation requires repository write access (to trigger `workflow_dispatch` or push a branch) or compromise of the shared Depot project or persistent build host. Because a poisoned `CARGO_TARGET_DIR` cache in Depot project `gjcfbjk8wc` persists into a subsequent tagged release build, the impact on release artifacts is High while the write-access precondition keeps Likelihood Low.

Recommendation

Isolate release builds from `dry_run` / non-tag BuildKit caches (pass `--no-cache` or use a separate Depot project for tagged releases), scope or remove the `zizmor: ignore[cache-poisoning]` suppression, and verify signed release tags in `zakura-mainnet-deploy.yml` .

© 2026 Quorint

Low (12)

ZAK-16 Windows major upgrade abandons the old database

RATING	STATUS	TARGET	CATEGORY
☐ Low	Open	crates/zakura-state/src/service/finalized_state/disk_db.rs:1457-1585	Database and upgrade (Windows)

`DiskDb::try_reusing_previous_db_after_major_upgrade` (`disk_db.rs:1457-1585`) canonicalizes `old_path` (`disk_db.rs:1479-1484`) and passes that canonical path to `DB::list_cf` (`disk_db.rs:1510`). On Windows, `std::fs::canonicalize` returns an extended-length verbatim path (`\\?\`) where Win32 does not rewrite RocksDB's forward-slash joins to backslashes, so `DB::list_cf` fails and `.is_ok_and(|cf| !cf.is_empty())` silently maps the probe error to `old_db_exists = false`. The reuse guard `old_db_exists && !new_db_exists` (`disk_db.rs:1513`) then falls through to `None` (`disk_db.rs:1584`) without logging, leaving the previous database directory on disk and opening a fresh empty database (`create_if_missing(true)` at `disk_db.rs:1594`) that resyncs from genesis.

Recommendation. Pass the non-canonical `old_path` (or strip the `\\?\` prefix on Windows) when probing `DB::list_cf`, and log a warning whenever `old_path` exists on disk but the RocksDB column-family probe or `fs::rename` fails.

ZAK-17 invalidateblock drops siblings and can hang the checkpoint path

RATING	STATUS	TARGET	CATEGORY
☐ Low	Open	crates/zakura-state/src/service/non_finalized_state.rs:97-99, 446-490, service.rs:1382-1399, write.rs:1986, 2150, 2346-2353	Operator RPC

`NonFinalizedState::invalidate_block` (`non_finalized_state.rs:446-490`) removes every chain containing the hash (`chain_set.retain` at `:451-458` and `insert_with` at `:459-470`) but records only the highest-work chain from `find_chain`, and keys `self.invalidated_blocks: IndexMap<Height, Arc<Vec<ContextuallyVerifiedBlock>>>` (`:97-99`) by `Height` (`insert` at `:474-481`, noted by a `#9552 TODO` at `:472`). A second invalidation at the same height replaces the first record, allowing a re-delivered first sibling to pass the commit-time check (`:665-673`). Separately, `send_invalidate_block` (`service.rs:1382-1399`) refuses with `ProcessingCheckpointedBlocks` only when `non_finalized` is `None`, whereas the writer installs `Some(...)` before the checkpoint loop (`write.rs:1986`) and pushes non-control messages onto `deferred_non_finalized_messages` (`write.rs:2150, 2346-2353`), so an `invalidateblock` or `reconsiderblock` issued during checkpoint sync blocks until that phase ends and then fails with `BlockNotFound`.

Recommendation. Key `invalidated_blocks` by `block::Hash` and record all removed chains so sibling invalidations are preserved, and reject `InvalidateBlock` / `ReconsiderBlock` immediately when `block_write_sender.finalized` is `Some(_)` (matching the `(finalized: None, non_finalized: Some(sender))` guard used by `send_precious_block` at `service.rs:1426-1435` in `454f0767`).

© 2026 Quorint

ZAK-18 Rollback uses unbounded memory and can wipe the backup

RATING	STATUS	TARGET	CATEGORY
☐ Low	Open	crates/zakura-state/src/service/finalized_state/zakura_db/rollback.rs:293-328, 472-515	Rollback and backup

In the offline `zakura-rollback-state` tool, `prepare_rollback` (`rollback.rs:472-515`) accumulates delete operations for every rolled-back height into a single in-memory RocksDB `DiskWriteBatch` (`rollback.rs:505-512`) as well as buffering `removed_blocks` even when `--backup-non-finalized` (`keep`) is false. In addition, because `clear_backup_dir` runs before `db.write_batch(prepared.batch)` (`rollback.rs:306, 318, 326`), a failed rollback leaves the finalized database intact at its prior tip (since the RocksDB batch write is atomic) but loses the previous non-finalized backup.

Recommendation. Chunk `DiskWriteBatch` commits across bounded height ranges so rollback memory is bounded even when `keep` is false, omit `removed_blocks` buffering when `keep` is false, and rotate the backup directory only after `db.write_batch` succeeds.

ZAK-19 Orphaned-block transactions are not restored after a reorg

RATING	STATUS	TARGET	CATEGORY
☐ Low	Open	crates/zakura-state/src/service/chain_tip.rs:680-688, 796-801, zakurad/src/components/mempool.rs:532-574, 701-712	Reorg handling

When a block is mined (`TipAction::Grow`), the mempool removes its mined transactions without retaining a copy (`mempool.rs:701-712`). If a subsequent reorganization (`TipAction::Reset`) orphans that block and its transactions were not also included in the winning branch, the mempool only re-verifies transactions still in `active_state` (`mempool.rs:532-574`), so transactions from the orphaned block are not returned to the mempool.

Recommendation. Pass non-coinbase transactions from orphaned blocks in `TipAction::Reset` (or retain recently mined transactions across the finality window) so non-conflicting transactions are re-queued into the mempool after a reorg.

ZAK-20 Gossiped blocks at an unexpected height are dropped without scoring

RATING	STATUS	TARGET	CATEGORY
☐ Low	Open	crates/zakurad/src/components/auth_download_height.rs:16-32, inbound/downloads.rs:760-806	Peer scoring

On the legacy inbound block gossip path, `tip_child_mismatch` (`auth_download_height.rs:16-32`) checks the coinbase height only when `block.header.previous_block_hash == tip_hash`. Off-height blocks that exceed the lookahead window or fall behind the tip are dropped in `inbound/downloads.rs:760-806` with plain string errors and `None` peer address, applying no misbehavior score to the sender.

Recommendation. Attach the advertising peer's `PeerSocketAddr` to off-height gossiped block drops and apply a small misbehavior penalty so peers flooding off-height blocks are deprioritized.

© 2026 Quorint

ZAK-21 The mempool crawler bypasses the per-peer request cap

RATING	STATUS	TARGET	CATEGORY
☐ Low	Open	crates/zakurad/src/components/mempool/downloads.rs:128, 136-138, 405-432, mempool.rs:948-952	Rate limiting

`Downloads::download_if_needed_and_verify` enforces `MAX_INBOUND_CONCURRENCY_PER_PEER = 5` only when `source` is `Some(peer_addr)` (`downloads.rs:416-431`). When the mempool crawler fans out transaction inventory (`mempool.rs:948-952`), it passes `source: None`, bypassing the per-peer cap (though the global queue remains bounded by `MAX_INBOUND_CONCURRENCY = 500` at `downloads.rs:128`).

Recommendation. Tag crawler-initiated transaction downloads with the peer that supplied the inventory or enforce a separate concurrency budget for crawler requests.

ZAK-22 Sapling batch-verification fallback amplifies CPU for a crafted batch

RATING	STATUS	TARGET	CATEGORY
☐ Low	Open	crates/zakura-consensus/src/primitives.rs:21, src/primitives/sapling.rs:176-242, 248-276, 299, crates/tower-fallback/src/future.rs:86-97	CPU amplification

Sapling spend and output proofs are batch-verified in batches of up to `MAX_BATCH_SIZE = 64` (`primitives.rs:21`, `sapling.rs:176-242, 299`). When a single item in a batch is invalid, `tower_fallback::Fallback` (`future.rs:86-97`) re-verifies every queued item in that batch individually via the unbatched fallback service (`sapling.rs:248-276`), amplifying verification CPU for that batch by a bounded constant factor.

Recommendation. Use bisecting batch verification on failure or attribute and score the peer that supplied the invalid transaction or block.

ZAK-23 v2 streams have no message tables (per-kind accounting)

RATING	STATUS	TARGET	CATEGORY
☐ Low	Open	crates/zakura-network/src/zakura/transport/service.rs:135-144	v2 stack

`Stream::PERSISTENT` (`transport/service.rs:135-144`) sets `messages: None`, and all four production v2 persistent stream declarations (control, discovery, header sync, and block sync) inherit it without populating per-message-type wire tables, relying on stream-wide frame size and rate limits rather than per-kind frame validation at the transport layer.

Recommendation. Populate explicit `MessageDecl` tables for each v2 persistent stream so message-type allowlists and per-kind payload ceilings are enforced before application decoding.

© 2026 Quorint

ZAK-24 QUIC receive windows have no aggregate memory budget

RATING	STATUS	TARGET	CATEGORY
☐ Low	Open	crates/zakura-network/src/zakura/handler.rs:79, 157-161, 513-515	v2 transport

The v2 QUIC transport configures a 16 MiB stream receive window (`DEFAULT_ZAKURA_STREAM_RECEIVE_WINDOW = 16 MiB`), a 32 MiB connection receive window (`DEFAULT_ZAKURA_RECEIVE_WINDOW = 32 MiB`), and a 32 MiB send window (`DEFAULT_ZAKURA_SEND_WINDOW = 32 MiB` at `handler.rs:157-161, 513-515`) per connection across up to `DEFAULT_ZAKURA_MAX_CONNECTIONS = 256` peers (`handler.rs:79`), with no aggregate node-wide QUIC buffer ceiling (up to 8 GiB worst-case across 256 connections if all windows are simultaneously filled).

Recommendation. Reduce the default per-connection QUIC receive/send windows for non-bulk streams or impose an aggregate memory budget across active v2 connections.

ZAK-25 Discovery records do not prove address ownership

RATING	STATUS	TARGET	CATEGORY
☐ Low	Open	crates/zakura-network/src/zakura/discovery/protocol.rs:3653-3721	v2 discovery

As documented in `protocol.rs:3698-3699` (“A signed `ZakuraNodeRecord` proves control of the node key, not ownership of the advertised `direct_addrs`”), any Ed25519 key holder can publish signed discovery records advertising third-party public `SocketAddr` endpoints, causing other v2 nodes to queue outbound QUIC dial probes to those addresses (mitigated by bogon filtering, a cap of 8 addresses per record, and a 4-concurrent-dial limit).

Recommendation. Promote advertised `direct_addrs` to the routable peer book only after a successful bidirectional QUIC handshake (with QUIC address validation) confirms that the key holder controls the address.

ZAK-26 A retryable verification error has no give-up bound

RATING	STATUS	TARGET	CATEGORY
☐ Low	Open	crates/zakura-network/src/zakura/header_sync/scheduler/retry.rs:10-12, 194-206, zakura-consensus/src/block.rs:182-190, 403-411	Error handling

In `header_sync/scheduler/retry.rs:10-12, 194-206`, after a retryable episode reaches `ALARM_ATTEMPTS = 10` or `ALARM_AFTER = 10 min`, the scheduler sets `self.alarmed = true` and continues probing every `ALARM_PROBE_INTERVAL = 10 min` indefinitely with no terminal cap. On the block-verification side, copy-level coinbase height errors (`MissingHeight` / `MaxHeight`, `block.rs:182-190, 403-411`) run before the Merkle root check and classify as `Retryable`.

Recommendation. Check `block.merkle_root()` before returning `MissingHeight` / `MaxHeight`, and add a configurable terminal give-up bound or operator circuit-breaker to alarmed retry episodes.

© 2026 Quorint

ZAK-27 Inherited legacy P2P serving limits may be loose for Zakura's model

RATING	STATUS	TARGET	CATEGORY
☐ Low	Open	crates/zakurad/src/components/inbound.rs:80, 90, 678, 764-779, sync.rs:107, 2963-2972, zakura-network/src/protocol/external/inv.rs:190, 201	Serving limits

Two bounded legacy P2P serving behaviors are inherited from Zebra: (1) while block `getdata` serving is capped at `GETDATA_MAX_BLOCK_COUNT = 16` (`inbound.rs:90, 678`), transaction `getdata` (`inbound.rs:764-779`) looks up all decoded IDs (up to `MAX_INV_IN_RECEIVED_MESSAGE = 50,000` at `inv.rs:190`) in the mempool and returns an uncapped `notfound` list; and (2) a peer that advertises a block hash and then times out on `getdata` is treated as a transient failure (`is_poisoned_body` at `sync.rs:2963-2972`) and retried up to `TRANSIENT_BLOCK_DOWNLOAD_RETRY_LIMIT = 3` times (`sync.rs:107`, 20 s per window; rescued in 2–8 s on v2 at `block_sync/config.rs:77, 84`).

Recommendation. Clamp inbound transaction `getdata` serving to `MAX_TX_INV_IN_SENT_MESSAGE = 25,000` (`inv.rs:201`) and deprioritize a sole advertiser whose `getdata` times out.

Informational (8)

ZAK-28 Fork parser accepts a pre-NU6.3 v6 branch ID

RATING	STATUS	TARGET	CATEGORY
☐ Informational	Open	crates/zakura-chain/src/transaction/serialize.rs:831-836, 1194-1198, transaction.rs:1857-1872, zakura-primitives 2.2.0 (transaction/mod.rs:808-815, 864-881)	Parser leniency

Native `zakura-chain` deserialization and serialization reject V6 transactions carrying a pre-NU6.3 `ConsensusBranchId` (`serialize.rs:831-836, 1194-1198`), whereas `read_v6` / `read_header_fragment` in `zakura-primitives 2.2.0` (`3.0.0` in `cd103f69`; `transaction/mod.rs:808-815, 864-881`) checks only `BranchId::try_from` and never calls `TxVersion::valid_in_branch`. No divergence is reachable today because `to_librustzcash` (`transaction.rs:1857-1872`) is the sole production caller of `Transaction::read` and re-serializes an already-accepted native transaction first.

Recommendation. Enforce `TxVersion::valid_in_branch` inside `zakura-primitives` (`read_header_fragment`) so any future direct caller cannot parse a pre-NU6.3 V6 transaction.

© 2026 Quorint

ZAK-29 Invariant sites panic on shared (non-per-peer) paths

RATING	STATUS	TARGET	CATEGORY
☐ Informational	Open	zakura-script (lib.rs:438-442), zakura-state (service/check.rs:299-331), zakura-chain (value_balance.rs:443-465, amount.rs:80-87, parameters/network/subsidy.rs:1053-1055, history_tree.rs:305-311), zakura-consensus (block/subsidy.rs:39), zakura-network (zakura/block_sync/reorder.rs:354-361, zakura/block_sync/peer_registry.rs:569-576), zakurad (mempool/storage/eviction_list.rs:63-66)	Panic containment

Thirteen invariant sites across ten functions on shared consensus, state, network, and mempool paths use `assert!`, `.expect(...)`, or `panic!(...)` inside node-wide tasks. Twelve are latent behind verified upstream guards, while one— `assert!(index > 0 && index <= num_addresses)` in `zakura-consensus/src/block/subsidy.rs:39` —is reachable from operator configuration on a custom or Regtest network with too few funding-stream addresses (`testnet.rs:347-353, 1522-1527`).

Recommendation. Convert shared-path `assert!` / `expect` / `panic!` invariant checks to typed `Result` errors and run `check_funding_stream_address_period` on Regtest configurations at startup.

ZAK-30 Checkpoint commit has no park path and flags VCT state early

RATING	STATUS	TARGET	CATEGORY
☐ Informational	Open	crates/zakura-state/src/service/write.rs:738-747, 2583-2588, 2703-2771, error.rs:352-356, finalized_state.rs:1185-1197	Commit symmetry

On the checkpoint commit path with a header chain attached, `commit_checkpoint_finalized` (`write.rs:738-747, 2583-2588`) maps `Stale` and `ResourceStalled` header-chain store outcomes to `CommitBlockError::HeaderChainError`, for which `vct_retryable_height` returns `None` (`error.rs:352-356`), resetting the checkpoint queue instead of parking (`write.rs:2703-2771`). In addition, `finalized_state.rs:1185-1197` updates the in-memory frozen-frontier flag (`vct.rs:569-579`) before `write_block_with` (`finalized_state.rs:1366-1374`) runs.

Recommendation. Update the in-memory frozen-frontier flag only after the database write succeeds, and handle retryable header-chain store stalls symmetrically across checkpoint and semantic commits.

© 2026 Quorint

ZAK-31 Some tree getters return empty trees instead of explicit absence

RATING	STATUS	TARGET	CATEGORY
☐ Informational	Open	crates/zakura-state/src/service/finalized_state/zakura_db/shielded.rs:221-225, 269-286, 414-428, 557-572, disk_format/upgrade.rs:625, 757-786	API clarity

When the finalized tip is inside the VCT fast-sync absent band `[U, H)` (`zakura_db/block.rs:729-735`), `sapling_tree_for_tip`, `orchard_tree_for_tip`, and `ironwood_tree_for_tip` (`shielded.rs:269-286, 414-428, 557-572`) `assert!(self.vct_tree_absent(height))` and return `Default::default()` (an empty tree) instead of an explicit absence error, unlike the Sprout tip getter which fails closed with `MissingSproutTipTree` (`shielded.rs:221-225`). On a band database, startup `format_validity_checks_detailed` (`disk_format/upgrade.rs:625, 757-786`) also reads these empty substitutes and succeeds only because incomplete subtree checks short-circuit and the empty Ironwood root equals the genesis anchor.

Recommendation. Return an explicit `Option` or typed enum from the tip tree getters so callers distinguish between a genuine pre-activation empty tree and a VCT-absent frontier, and skip tip-tree format checks when `vct_tree_absent(height)` holds.

ZAK-32 Block-template builder does not check transaction versions

RATING	STATUS	TARGET	CATEGORY
☐ Informational	Open	zakura-rpc (crates/zakura-rpc/src/methods/types/get_block_template/zip317.rs:397-424)	Template builder

`Summary::try_add` in `crates/zakura-rpc/src/methods/types/get_block_template/zip317.rs:397-424` checks block size, sigops, unpaid actions, and ZIP-218 shielded limits, but never checks whether a candidate transaction's version is valid for the template height's network upgrade. It is harmless today because `ChainTipChange` resets and re-verifies the mempool at upgrade activation (`chain_tip.rs:680-688`; `mempool.rs:532-571`).

Recommendation. Check each candidate transaction's version against the template height's network upgrade inside `try_add`.

© 2026 Quorint

ZAK-33 Redelivered auxiliary record fails a header batch

RATING	STATUS	TARGET	CATEGORY
☐ Informational	Open	zakura-header-chain (transition/planner/event_effects/header_admission.rs:260-269), zakurad (header_sync_driver.rs:471-482, 656-663), zakura-state (service/write/vct_write_retry.rs:340-348)	Header batch

When a stored auxiliary record's outcome is refined by integrated verification, a subsequent redelivery under the same `delivery_id` no longer compares equal in `event_effects/header_admission.rs:260-269` (because `without_scheduling_body_size()` does not strip `outcome`) and fails the whole header batch with `ReplayConflict`. This is not peer-reachable today: the inbound `delivery_id` hashes the source, session ID, request ID, and header hash (`header_sync_driver.rs:471-482, 656-663`), and VCT repair advances a strictly monotonic generation on each retry (`vct_write_retry.rs:340-348`).

Recommendation. Compare redelivered auxiliary records without their verification outcome (or scope `ReplayConflict` to the offending record).

ZAK-34 Orchard proof-size rule differs between native and fork parsers before NU6.2

RATING	STATUS	TARGET	CATEGORY
☐ Informational	Open	zakura-chain (transaction/serialize.rs:533-535, transaction/tests/vectors.rs:2140-2201), zakura-consensus (transaction/check.rs:217-231)	Parser consistency

Native parsing and semantic checks enforce canonical Orchard proof sizes across all transaction versions and network upgrades (`zakura-chain/src/transaction/serialize.rs:533-535`; `zakura-consensus/src/transaction/check.rs:217-231`), whereas the `zakura-orchard 2.2.0` (`3.0.0` in `cd103f69`) fork exempts the pre-NU6.2 `InsecureV1` protocol version (`bundle.rs:185-187`), as documented by `v5_orchard_proof_size_is_always_enforced_during_deserialization` (`vectors.rs:2140-2201`). No non-canonical proof can be accepted; if a historical pre-NU6.2 block contained a non-canonical Orchard proof, native parsing would fail closed and stall sync at that height.

Recommendation. Confirm no historical pre-NU6.2 block carries a non-canonical Orchard proof, or align the native pre-NU6.2 rule with the fork.

© 2026 Quorint

ZAK-35 Unaudited GLV multi-exponentiation is live on the proof path; the dormant orbits feature has no CI tripwire

RATING	STATUS	TARGET	CATEGORY
☐ Informational	Open	zakura-consensus (primitives/halo2.rs:89-99) / zakura-pasta-curves (glv.rs:2178-2224, qa/supply-chain/config.toml:427, 462, .github/workflows/lint.yml:269-271)	Forked cryptography

Orchard and Ironwood proof verification runs the custom signed-Booth GLV multi-exponentiation backend in `zakura-pasta-curves 2.2.0` (`3.0.0` in `cd103f69`; `glv.rs:2178-2224`), which carries a supply-chain audit exemption pending a direct package audit (`qa/supply-chain/config.toml:427, 462`). A second `MultiexpPlan::Orbit` arm is gated behind `#[cfg(feature = "orbits")]` (`glv.rs:2188-2212`); while `orbits` is disabled in default builds (`halo2.rs:89-99`), no CI `cargo tree -e features` check (unlike the `arbitrary|proptest` gate at `.github/workflows/lint.yml:269-271`) currently prevents a dependency from enabling `orbits` via Cargo feature unification. First-principles verification of the Pallas and Vesta GLV lattice constants and scalar decomposition passed with zero failures.

Recommendation. Complete a dedicated cryptographic audit of the live GLV multi-exponentiation implementation in `zakura-pasta-curves`, and add a CI feature-graph check (alongside `.github/workflows/lint.yml:269-271`) asserting that the dormant `orbits` feature never resolves in `zakura-pasta-curves` or `zakura-halo2-proofs`.

© 2026 Quorint

Disclaimer and version history

6.1 Disclaimer

This report reflects a point-in-time security assessment of Zakura at the revisions and scope listed in Section 2. A security assessment is a time-boxed, best-effort review and cannot prove the absence of all vulnerabilities; subsequent changes to code, dependencies, configuration, or deployment parameters may introduce new issues. All dynamic verification was performed offline in isolated local harnesses without targeting production peers or operator infrastructure.

6.2 Version history

VERSION	DATE	DESCRIPTION
1.0	14 October 2026	Final assessment report delivered to Dev Ojha (Zakura).

ABOUT QUORINT

Quorint is an independent security research firm specializing in consensus-critical software: layer-one full nodes, zero-knowledge circuits, distributed consensus protocols, and applied cryptography. Learn more at quorint.xyz or reach us at contact@quorint.xyz.

© 2026 Quorint