// Security Assessment

10.29.2025 - 11.11.2025

Hedera Blockchain Network

Swirld Labs (now Hashgraph)

HALBORN

Hedera Blockchain Network - Swirld Labs (now Hashgraph)

Prepared by: HALBORN

Last Updated 01/14/2026

Date of Engagement: October 29th, 2025 - November 11th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	1	0	0	0	0

TABLE OF CONTENTS

1. Introduction

`Hashgraph` engaged Halborn to conduct a Penetration Test on their systems, beginning on `Oct 29th 2025` and ending on `Nov 11th 2025`. The security assessment was scoped to the web components that `Hashgraph` shared with the Halborn team. See the 'Scope' section for more details.

2. Assessment Summary

The `Halborn Team` were allocated four days for this engagement, during which one full-time security engineer — expert in blockchain, web, and API security — conducted a comprehensive audit of the in-scope webapp components related to `Hashgraph` systems. This engineer possesses advanced skills in penetration testing, web application security assessments, and an in-depth understanding of multiple blockchain protocols.

The primary objectives of this audit were to:

- Verify that each component performs its intended functions accurately and reliably.
- Identify and assess potential security issues within the application, particularly those that could impact the management and protection of digital assets.

Additional objectives included evaluating adherence to industry best practices, ensuring robust security measures are in place, and identifying opportunities for further strengthening the security posture.

All components behaved as expected across positive and negative tests, with one exception: the critical finding that allowed to take down the Consensus Node.

Submitting oversized topic messages **without** limiting chunks causes the SDK to auto-chunk into multiple transactions. Under sustained bursts, this increased inbound pressure and memory use on the Consensus Node. This behavior aligns with the repeatable `OutOfMemoryError: Java heap space` events observed in your logs and during our re-runs. There is no evidence of asset integrity or authorization issues, but the service-level impact is critical.

As mentioned, the JSON-RPC Relay and Mirror Node showed consistent, correct behavior in the tested scope. The Consensus Node enforced signature and transaction rules as expected across edge cases. The main risk is an availability weakness tied to high-volume, auto-chunked HCS submits.

Introducing chunk-aware throttling and back-pressure at ingress, combined with sensible per-client budgets/limits, should materially improve resilience without affecting legitimate workloads.

3. Scope & Environment

All the components were initially **self-hosted by Halborn on EC2 instance in AWS.**

The `SOLO version` that Halborn was requested to use was `v0.48.0`.

Assume that the EC2 instance public IP was `1.2.3.4`.

In-scope components:

- **JSON-RPC Relay** (Ethereum-compatible) — `http://1.2.3.4:7546/`
- **Mirror Node REST API** (read-only) — `http://1.2.3.4:8081/` (OpenAPI at `/api/v1/docs/`)
- **Consensus gRPC** — `1.2.3.4:50211`

The Consensus gRPC interactions needed to be exercised via Hedera SDK (**https://github.com/hiero-ledger/hiero-sdk-js**) per client guidance.

Out of scope: Smart contracts, mainnet/testnet/previewnet, infra hardening checks (DoS/rate-limit/CORS/TLS/headers) due to self-hosted environment.

After several days after the engagement started, Hashgraph team provided a new fresh environment for further testing. The new environment information was:

- Network Node IP: `136.111.199.66` (TCP Port `50211`)
- Mirror Node IP: `34.31.228.61` (Docs in **http://34.31.228.61/api/v1/docs/**)
- Explorer URL: **http://34.27.150.139**
- JSON RPC Relay IP: **http://34.46.38.34:7546**
- JSON RPC Websockets IP: `136.112.75.57`

In the SOLO local deployment, the following entities were used for several tests:

Hedera **ECDSA Alias** accounts pre-provisioned by Solo (EVM-compatible). Primary keys used:

- **From:** 0.0.1012 — `0x105d…1524`
- **To:** 0.0.1013 — `0x2e1d…03e7`
- Confirmed chainId — `298` (`0x12a`)

For the testing scripts, the payer was `0.0.2`, with Private Key "`302e020<redacted>7f3087137`".

4. Test Approach And Methodology

`Halborn` combined manual and automated security testing to strike the right balance between speed, thoroughness, and precision within the scope of the penetration test. Manual techniques were employed to reveal nuanced logical, procedural, and implementation-level flaws, while automated tools broadened the assessment's reach—rapidly pinpointing common vulnerabilities across the entire solution.

Throughout the engagement, we progressed through, among the following —but not limited to— phases, leveraging both targeted tools and bespoke techniques:

- **Content & Functionality Mapping**
  Cataloging every feature and endpoint exposed by the application.
- **Sensitive Data Exposure**
  Hunting for leaks of critical or private information.
- **Business-Logic Testing**
  Uncovering flaws in workflows, transaction flows, and access controls.
- **Access Control Assessment**
  Verifying correct enforcement of permissions and role-based restrictions.
- **Input Validation & Handling**
  Ensuring all user inputs are properly sanitized and parsed.
- **Fuzzing & Parameter Injection**
  Applying randomized and structured payloads—SQL, JSON, HTML, command-line, directory path injections—to provoke unexpected behavior.

We executed a **grey-box interface robustness assessment** focused on:

- **Positive controls** (well-formed, expected happy paths),
- **Negative/abuse vectors** (malformed, boundary, and semantic error cases),
- **Logical security checks** (chain-id mismatch behavior, creation gas validation, oversized payload handling),
- **Replay & nonce sequencing** (replays, nonce-too-low paths).

The approach emphasizes **deterministic JSON-RPC behavior**, **error hygiene** (no 5xx, no stack traces), and **fault isolation** (input validation vs. backend failure). All traffic was captured through a Burp proxy to preserve full evidence.

Over ten days, Halborn reviewed the off-chain components that support the Hashgraph deployment: the **Consensus Node (gRPC)**, the **Mirror Node (REST)**, and the **JSON-RPC Relay**. Halborn combined manual testing with a scripted harness to exercise both normal flows and edge cases. Every scenario was run in two modes—first with a system-like payer for operational parity, then with a neutral user account—to rule out environment-specific behavior.

Consensus Node (gRPC)

We validated account lifecycle (create/update/delete), HBAR transfers, and a wide set of failure conditions: missing or wrong signatures (including ED25519/ECDSA cross-type attempts), invalid transaction windows, node routing to unknown IDs, payer/account existence, non-zero-sum transfers, tiny fees, insufficient balances, partial multisig, and reuse of deleted accounts. We also covered Hedera Consensus Service (HCS) topics: submits with and without `submitKey`, and large messages with and without explicit chunk limits. Results were cross-checked against Mirror/Relay visibility.

Mirror Node (REST)

We verified account, transaction, and node endpoints for correctness and alignment with gRPC results. Responses consistently matched consensus outcomes. Several tests were carried out, targeting per-endpoint parameter fuzzing (IDs, timestamps, topics, pagination, sort), injection separators, and boundary values, also including parameter tampering and type/format fuzz (e.g., integer overflow-ish strings, hex form variants, RFC3339 timestamp oddities).

JSON-RPC Relay.

We generated and executed **100+** distinct transactions and raw payload mutations covering:

- **Transaction types:** Legacy (type 0), Access-List (type 1), EIP-1559 (type 2).
- **Nonce logic:** sequences (`t2-seq-0..19`), replays of exact raw, future-gap nonces (+1, +100).
- **Fee/gas edge cases:** `maxPriorityFeePerGas`/`maxFeePerGas` at 0, inverted (`prio > max`), very low/high `maxFeePerGas`, `gasLimit` ∈ {0,1, very large}.
- **Chain ID variants:** `{0, 1, 296, 298}`.
- **Address/data anomalies:** `to` absent (contract creation), short/long/non-hex `to`, odd-length/invalid-hex/large `data`.
- **Raw-level mutations:** strip `0x`, odd length, insert “GG”, truncate, large append (~10 KB), case flips.

Under background activity, Relay behavior remained stable and in sync with Mirror and gRPC receipts.

5. RISK METHODOLOGY

Halborn assesses the severity of findings using either the Common Vulnerability Scoring System (CVSS) framework or the Impact/Likelihood Risk scale, depending on the engagement. CVSS is an industry standard framework for communicating characteristics and severity of vulnerabilities in software. Details can be found in the **CVSS Specification Document** published by F.I.R.S.T.

Vulnerabilities or issues observed by Halborn scored on the Impact/Likelihood Risk scale are measured by the LIKELIHOOD of a security incident and the IMPACT should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of **10** to **1** with **10** being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

- **10** - CRITICAL
- **9 - 8** - HIGH
- **7 - 6** - MEDIUM
- **5 - 4** - LOW
- **3 - 1** - VERY LOW AND INFORMATIONAL

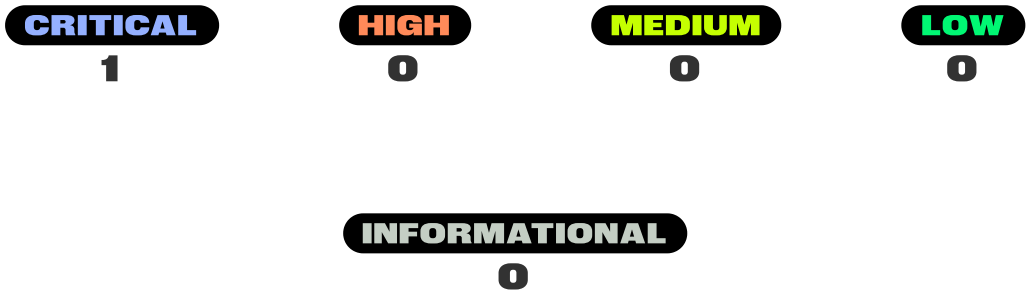
6. SCOPE

REMEDIATION COMMIT ID:

- https://github.com/hiero-ledger/hiero-consensus-node/issues/22295

Out-of-Scope: New features/implementations after the remediation commit IDs.

7. ASSESSMENT SUMMARY & FINDINGS OVERVIEW



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
CONSENSUS NODE - DENIAL OF SERVICE	CRITICAL	SOLVED - 12/19/2025

8. FINDINGS & TECH DETAILS

8.1 CONSENSUS NODE - DENIAL OF SERVICE

// CRITICAL

Description

During negative and load-style testing of Hedera Consensus Service (HCS), it was observed that submitting large topic messages **without disabling SDK auto-chunking** can generate a high volume of chunked transactions in a short window. When repeated across multiple sizes and runs, this induces sustained inbound pressure on the consensus node's gRPC transport and state snapshotting components (MerkleDB), culminating in `java.lang.OutOfMemoryError: Java heap space` and node unresponsiveness ("lights on, no one home").
Key observations from the affected node:

- Recurrent `OutOfMemoryError` in Netty gRPC transport (`NettyServerTransport notifyTerminated`) and background tasks (e.g., `idle-timeout-task`, JVM pause detector).
- Concurrent OOMs while creating **periodic signed state snapshots**:
  - `MerkleDbDataSource: Snapshot keyToPath failed`
  - `MerkleDbDataSource: Snapshot pathToDiskLocationInternalNodes failed`
- Temporary platform status oscillations (ACTIVE → CHECKING → ACTIVE) followed by the process becoming effectively unresponsive to clients; SDK errors report **"All nodes are unhealthy"**.

Impact

The impact is CRITICAL, as this issue may cause a complete denial of service in the HCS.

- **Consensus node availability loss.** The node exhausts heap and stops servicing requests; clients receive "all nodes unhealthy" or timeouts.
- **Network-level degradation.** While the issue manifests on a single node in this setup, similar pressure on a multi-node network could degrade service, increase consensus latency, or trigger failover behavior.
- **Operational instability.** OOMs occurring during **signed state snapshot** creation (MerkleDB) increase recovery time and risk of repeated instability on restart without remediation.

Proof of Concept

Listed below, there are evidences that show how the Consensus Node was not responding after the attack:

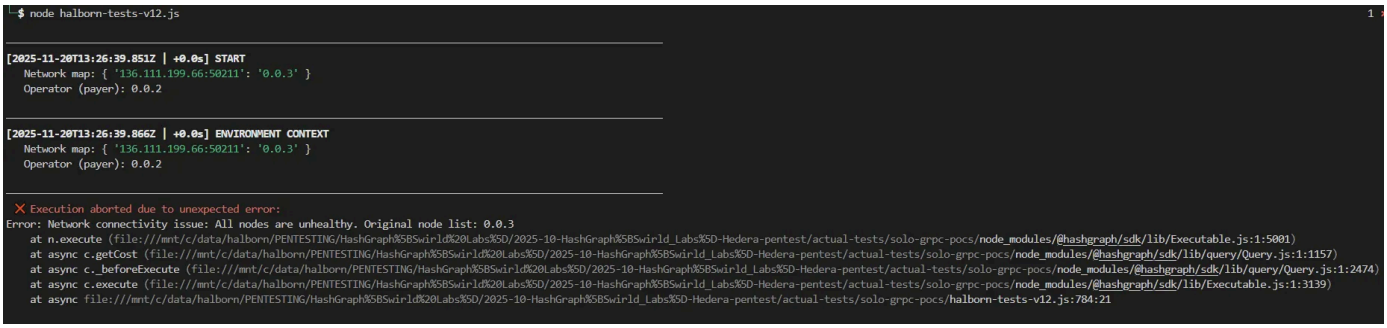
- Evidence 1: tried to re-run the test script and got error: `All nodes are unhealthy. Original node list: 0.0.3`

```
$ node halborn-tests-v12.js
```

```
[2025-11-20T13:26:39.851Z | +0.0s] START
   Network map: { '136.111.199.66:50211': '0.0.3' }
   Operator (payer): 0.0.2

[2025-11-20T13:26:39.866Z | +0.0s] ENVIRONMENT CONTEXT
   Network map: { '136.111.199.66:50211': '0.0.3' }
   Operator (payer): 0.0.2

❌ Execution aborted due to unexpected error:
Error: Network connectivity issue: All nodes are unhealthy. Original node list: 0.0.3
    at n.execute (file:///mnt/c/data/halborn/PENTESTING/HashGraph%5BSwirld%20Labs%5D/2025-10-H
    at async c.getCost (file:///mnt/c/data/halborn/PENTESTING/HashGraph%5BSwirld%20Labs%5D/202
    at async c._beforeExecute (file:///mnt/c/data/halborn/PENTESTING/HashGraph%5BSwirld%20Labs
    at async c.execute (file:///mnt/c/data/halborn/PENTESTING/HashGraph%5BSwirld%20Labs%5D/202
    at async file:///mnt/c/data/halborn/PENTESTING/HashGraph%5BSwirld%20Labs%5D/2025-10-HashGr
```



- Evidence 2: Same error as above, but this time while using a “Create Account” script from `hiero-sdk-js` SDK “examples” folder:

Score

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:N/I:L/A:H (9.3)

Recommendation

The following recommendations take into account the feedback that `Hashgraph` shared, including error logs and `Hashgraph` team messages.

High-Level: What To Do

- **Throttle and budget** HCS chunks (per account/topic and globally).
- **Apply back-pressure** in gRPC/Netty to bound inflight requests and buffers.
- **Constrain auto-chunking** (lower max chunks; require explicit opt-in for large messages).
- **Harden operations** (right-size heap/GC; snapshot under load must be memory-bounded).
- **Continuously test** with mixed traffic + auto-chunked bursts and monitor GC/queues.

Concrete Near-Term Steps

- **Ingress limits:** Enforce per-account/topic TPS and **bytes/sec** for `TopicMessageSubmit`, plus a **cluster-wide chunk budget**; reject or defer past the limit.
- **Transport back-pressure:** Configure Netty high/low watermarks, cap concurrent streams, and bound message/frame sizes; add a **circuit breaker** when GC pauses or queue depths exceed thresholds.
- **Auto-chunking policy:** Reduce **max chunks/message** and/or require an explicit flag beyond a small threshold; consider **progressive fees** for multi-chunk messages.
- **Operational guardrails:** Increase heap and tune GC (e.g., G1/ZGC) on dev/test nodes; ensure snapshotting uses **streaming/iterative** paths with bounded memory.
- **Validation:** Add soak tests that replay large auto-chunked traffic during snapshots; alert on chunk rate, inflight buffers, and GC pauses.

Remediation Comment

SOLVED: The **Hashgraph team** addressed this issue.

Remediation Hash

https://github.com/hiero-ledger/hiero-consensus-node/issues/22295
