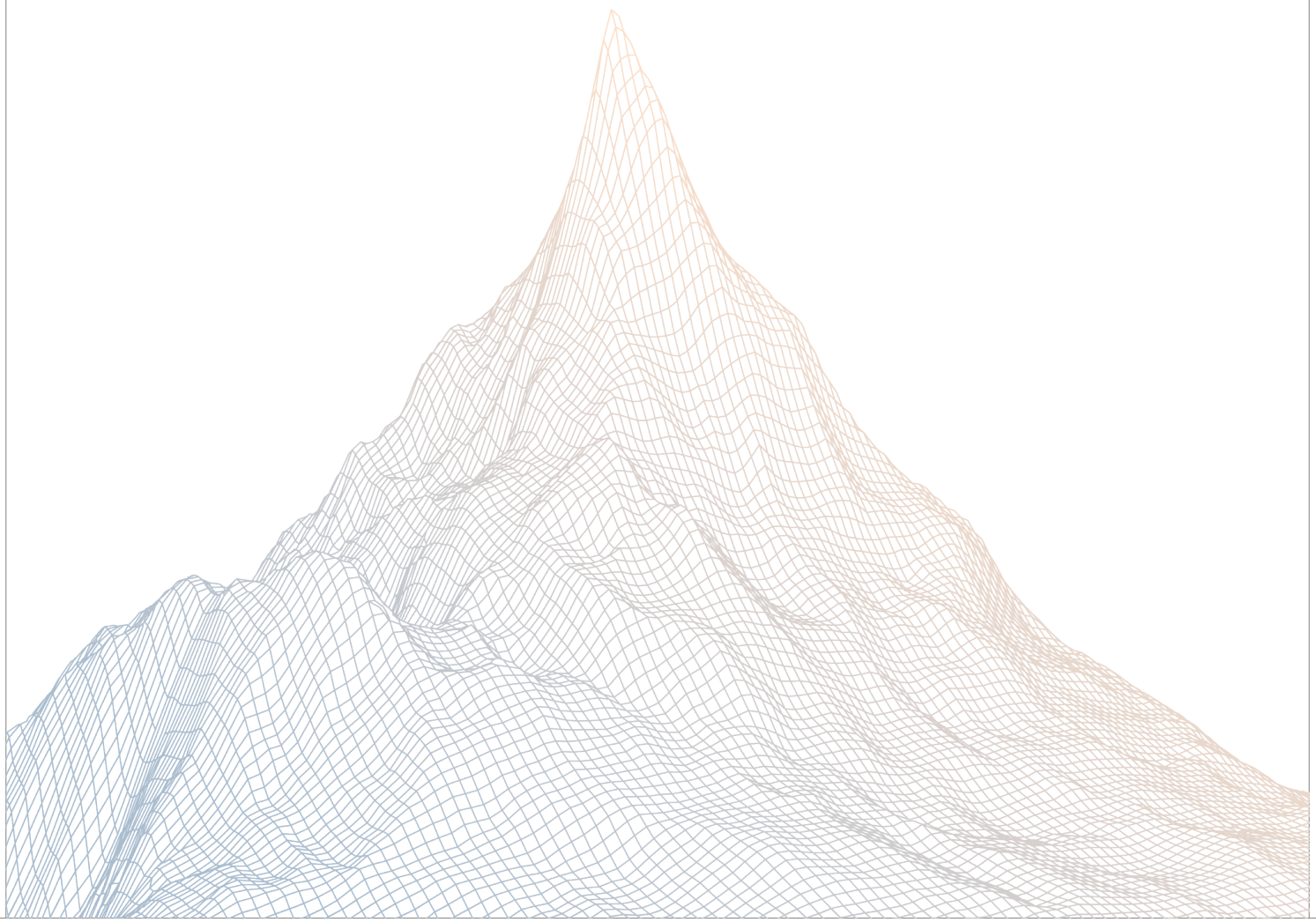
Zenith

Plasma

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: August 29th to September 11th, 2025

AUDITED BY: Bernd
Christian Vari

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

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

2

Executive Summary

2.1 About Plasma

Plasma is a layer 1 blockchain designed for global stablecoin payments.

Plasma is secured by PlasmaBFT, a high-performance implementation of Fast HotStuff written in Rust. It combines the safety of Byzantine Fault Tolerant (BFT) consensus with low-latency finality, enabling the high throughput and deterministic guarantees required for stablecoin-scale applications.

2.2 Scope

The engagement involved a review of the following targets:

Target	plasma-consensus
Repository	https://github.com/PlasmaLaboratories/plasma-consensus.git
Commit Hash	`99b8e16948a2d388da28149f281a0b583075bf80`

Files	`consensus`
	`node`
	`networking`
	`block_builder`
	`execution_proxy`

2.3 Audit Timeline

August 29, 2025	Audit start
September 11, 2025	Audit end
October 24, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	2
Low Risk	11
Informational	9
Total Issues	**22**

3

Findings Summary

ID	Description	Status
M-1	Node panics when the maximum number of QC voters is exceeded	Resolved
M-2	Node synchronization can be stalled by proposing a block with a high block height that refers to a non-existent block	Resolved
L-1	Missing HTTP client timeouts allow API handler to hang	Resolved
L-2	Rigid accept parsing breaks HTTP content negotiation and causes erroneous 406s	Resolved
L-3	Missing TLS certificates prevent clients from authenticating API and metrics servers	Acknowledged
L-4	Missing HTTP timeouts and request limits expose API and metrics servers to Slowloris like attacks and floods	Acknowledged
L-5	Block re-broadcast by byzantine primary enables network wide cpu denial of service.	Resolved
L-6	NewView validation enables targeted cpu denial of service against primaries	Resolved
L-7	Fatal termination of SyncManager on stream closure	Resolved
L-8	Length field width mismatch causes spurious empty frames in block decoding	Resolved
L-9	Insecure file permissions on Windows enable identity key disclosure	Acknowledged
L-10	Insecure transport schemes allow JWT disclosure and engine impersonation	Acknowledged
L-11	Multi-chunk block catch-up stalls after first chunk	Resolved
I-1	Mismatched block size limits	Resolved
I-2	Trusted peers bypass connection limits without safeguards enabling denial of service	Acknowledged
I-3	Unused enable_peer_scoring flag	Acknowledged

ID	Description	Status
I-4	Incorrect error variant returned for oversized block envelopes	Resolved
I-5	Outdated comment for MAX_NETWORK_RAW_BLOCK_SIZE consensus constant	Resolved
I-6	Unchecked UnicastRequest data length in UnicastRequest::into_request_rpc	Resolved
I-7	Unvalidated QC proposer_index during block validation	Resolved
I-8	Unbounded timeout backoff exponent enables panic and denial of service	Resolved
I-9	Network manager continuously attempts to reconnect blacklisted peers	Acknowledged

4

Findings

4.1 Medium Risk

A total of 2 medium risk findings were identified.

[M-1] Node panics when the maximum number of QC voters is exceeded

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

Target

- crates/types/src/domain/consensus/block.rs#L105
- crates/types/src/domain/consensus/block.rs#L175
- crates/types/src/domain/consensus/block.rs#L236

Description:

The `ListOfVoters::tree_hash_root()` function, which is used during block processing, can panic if the number of votes in a block's Quorum Certificate (QC) exceeds `MAX_VOTERS_COUNT` (currently set to 200). Specifically, `tree_hash_root` calls `try_tree_hash_root`, and then uses `.expect()` on the result. If `try_tree_hash_root` returns an error (which it does when the voter list is too long), the node will panic and crash.

A proposer can craft a block containing a QC with more than 200 votes. When this block is gossiped to other nodes in the network, they will attempt to validate it. The validation process involves calculating the hash tree root of the QC's votes, which triggers the panic.

This can be used by a dishonest validator to selectively crash nodes or cause a widespread denial of service across the network by propagating a malicious block. This panic will be thrown even before other validation checks on the votes, such as checking for duplicates.

Note that `.expect()` is also used in other instances of `tree_hash_root()`, causing a similar issue.

Recommendations:

Consider replacing the `.expect()` with proper error handling in `tree_hash_root()`. The error should be propagated up to the validation function, allowing the node to reject the invalid block gracefully without crashing. This would turn a fatal panic into a regular

validation error.

Plasma: Resolved with PR-1290.

Zenith: Verified.

[M-2] Node synchronization can be stalled by proposing a block with a high block height that refers to a non-existent block

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

Target

- crates/consensus/src/validation/validator_block.rs#L678
- crates/node/src/domain/block_streamer.rs#L71-L82

Description:

During block validation, the `validate_parent_block_and_state_exist()` function is called with the `block_height` from a new, not-yet-fully-validated block. While the verified QC's block hash is used to retrieve the parent block, this height is used to determine the upper bound of a block range for the catch-up mechanism if the parent block of the new block is not found in the local store.

A malicious validator can craft a block with a valid Quorum Certificate (QC) but an arbitrarily high `block_height`. When another node receives this block, it will attempt to validate it. If the parent block is missing, the node initiates a catch-up process, requesting a block range up to the exaggerated height from its peers.

When a peer receives this request for a range ending far in the future, its `store.block_stream()` function will fail, as the requested block does not exist. The `block_streamer` on the peer node is designed to only log this error and sends no response back to the initiator.

```
71: match store.block_stream(range) {
72:     Ok(stream) ⇒ {
73:         debug!(
74:             "Built stream for range {:?}. Processing stream request.",
75:             range
76:         );
77:         process_stream_request(peer, stream, sink);
78:     }
79:     Err(e) ⇒ {
80:         error!("Error getting block range: {:?}", e);
81:     }
```

```
82: }
```

This causes the requesting node to time out and enter an indefinite retry loop.

While stuck in this loop, the node's catch-up state is locked in the `WaitingForStream` phase, preventing it from processing any new, legitimate catch-up requests. This effectively stalls the node and prevents it from synchronizing with the network, leading to a Denial of Service.

Recommendations:

Consider taking the following actions to remediate this issue:

1. In the `validate_block_internal` function, when calling `validate_parent_block_and_state_exist`, use the block height from the verified QC (`verified_qc.qc().block_height()`) to determine the target height for catch-up, instead of the unverified `new_block.block_height()`. This ensures that the catch-up mechanism is triggered for a valid and existing block height.
2. The error handling in the `block_streamer`'s `on_stream_request_arrived` function should be improved. When `store.block_stream()` fails, instead of silently logging the error, the node should send a specific failure response indicating the non-existent block. This allows the initiator to gracefully handle the failed attempt and prevents it from getting stuck in a timeout-retry loop.

Plasma: Resolved with PR-1327.

Zenith: Verified.

4.2 Low Risk

A total of 11 low risk findings were identified.

[L-1] Missing HTTP client timeouts allow API handler to hang

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

Target

- crates/node/src/api/handlers/api_get_peer_scores.rs#L48-L62

Description:

The API handler for peer scores uses `reqwest::get` without specifying any request or read timeouts. If the metrics endpoint becomes slow or unresponsive, the request can block indefinitely.

This causes the API route to hang, degrading responsiveness and potentially leading to denial-of-service (DoS) if multiple requests pile up.

Recommendations:

We recommend configuring explicit client-side timeouts for all HTTP requests.

Plasma: Resolved with PR-1194

Zenith: Verified.

[L-2] Rigid accept parsing breaks HTTP content negotiation and causes erroneous 406s

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

Target

- crates/node/src/api/handlers/api_get_block_by_id.rs#L135-L181

Description:

The block-by-id handler compares the `Accept` header to two exact strings (`application/json` and `application/octet-stream`).

It does not parse comma-separated media ranges or honor q-weights, rejects valid parameters (e.g., `application/json; charset=utf-8`), ignores wildcards (`/`, `application/`),and fails to match vendor types with the `+json` suffix (e.g., `application/vnd.api+json`).

Because media types are case-insensitive, literal comparisons can also misclassify differently cased values. The result is unexpected `406 Not Acceptable` responses for common clients (e.g., curl's default) and broken content negotiation, harming availability and interoperability.

Recommendations:

We recommend parsing `Accept` via `TypedHeader<headers::Accept>`.

Plasma: Resolved with PR-1321.

Zenith: Verified.

[L-3] Missing TLS certificates prevent clients from authenticating API and metrics servers

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

Target

- crates/node/src/api

Description:

The API and metrics servers are exposed over plain TCP without TLS. In this configuration, clients have no way to authenticate the server they connect to, since no certificate is presented.

This leaves them vulnerable to man-in-the-middle (MITM) attacks: a malicious intermediary could impersonate the node, intercept sensitive health/state data, forge API responses, or inject falsified metrics.

Recommendations:

We recommend enabling TLS with valid certificates to provide server authentication and encrypted transport:

- Reverse proxy termination: place a TLS-terminating proxy (e.g., nginx, Caddy, Envoy) in front of the services, configured with strong cipher suites and properly managed certificates.
- Native TLS: embed TLS directly into the service ensuring the node presents a certificate during handshakes.

Plasma: The client acknowledges the issue stating that the risk is acceptable.

[L-4] Missing HTTP timeouts and request limits expose API and metrics servers to Slowloris like attacks and floods

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

Target

- crates/node/src/api

Description:

The API and metrics servers are deployed with default settings and lack essential defensive controls.

Specifically:

- No per request, header-read, or body-read timeouts are enforced.
- No concurrency or rate limits are applied.
- Request body sizes are uncapped.

In this configuration, malicious actors can launch Slowloris like attacks by trickling headers or bodies to hold open server connections indefinitely, or overwhelm the node through floods of concurrent/high-rate requests, leading to resource exhaustion.

Recommendations:

We recommend hardening the HTTP layer as follows:

- Enforce server-side timeouts: header-read, body-read, keep-alive, and per-request execution deadlines.
- Cap request body sizes to a conservative upper bound.
- Introduce rate limiting.

Plasma: The client acknowledges the issue stating that the risk is acceptable as users are meant to expose these APIs only behind mitigating load balancers or other infrastructure.

[L-5] Block re-broadcast by byzantine primary enables network wide cpu denial of service.

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

Target

- crates/node/src/service/effect_runner.rs#L144-L151

Description:

When acting as primary, a validator proposes and gossips a `BlockEnvelope` to the committee and observers. A malicious primary can exploit this by repeatedly re-broadcasting the same valid-looking block within its leadership window.

Each received block forces every validator to execute the entire validation pipeline, including:

- Envelope checks (hash, height, view consistency, proposer-voter equality, single BLS signature).
- Heavy block validation (aggregate BLS verification of the QC or AggQC, quorum and duplicate voter checks, SSZ decoding, parent/state lookups, structural constraints such as consecutive views and monotonicity).
- Execution-layer validation via the engine client (`new_payload_v4` on Reth, RANDAO, execution-hash checks).
- New state construction and state-root verification.

Only after this expensive process does the state machine recognize the block as “old” and discard it. Because `gossipsub` IDs are tied to `libp2p` sequence numbers, re-sent blocks are not suppressed at the networking layer and are reprocessed in full by all validators.

The result is a network-wide resource-exhaustion vector: every repeated block forces all validators to spend CPU on aggregate BLS verification, perform store I/O and SSZ decoding, and invoke the execution engine to validate the payload. This does not directly target one specific future primary; instead it degrades everyone’s throughput and increases the probability of timeouts and view changes. In practice, sustained block spam by the current primary can make nodes process messages more slowly, so when their turn to be primary arrives, they may have less time to build/verify/propose and are more likely to hit a timeout, but the effect is diffuse rather than precisely targeted.

The attack has negligible cost to the adversary beyond signing and gossiping messages,

while incurring high verification costs for all victims.

Severity is low since at this stage the committee is trusted.

Recommendations:

We recommend deduping by block hash (and QC tuple) before performing QC/EL work.

Plasma: Resolved with PR-1394.

Zenith: Verified.

[L-6] `NewView` validation enables targeted cpu denial of service against primaries

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

Target

- crates/consensus/src/validation/validator_unicast.rs#L54-L124

Description:

The issue is a targeted CPU-denial-of-service against the primary (leader) for a view by abusing `NewView` validation. In this design, only the next-view primary processes `NewView` messages for view v.

Initial checks on `NewView` messages are inexpensive (committee membership, next-view primary role, single BLS signature validation). However, if the message is valid at this stage, the primary must perform aggregate Quorum Certificate (QC) verification, which involves reconstructing voter messages and validating an aggregate BLS signature. This step is significantly more expensive and scales with committee size.

A malicious committee member can exploit this by spamming multiple `NewView` messages for the same view. Since the protocol defers storage checks and lacks pre-verification deduplication or rate limiting, each spammed message forces the primary to repeat costly QC verification, even for invalid or duplicate QCs. The safeguard for not counting the same voter twice, in fact, only applies after verification, failing to prevent wasted computation.

Because leader selection is deterministic, the attacker can precisely time their spam against whichever node becomes primary. Sustained CPU exhaustion forces primaries to miss timeouts, rotating the primary and triggering repeated view changes and degrading throughput.

The cost to the adversary is zero, as they only require valid committee membership and a signing key; the protocol provides no slashing or economic deterrent. Larger committees further increase the per-message verification cost, amplifying the DoS effect.

As a consequence, a committee memeber can DoS any primary in order to let them fail their turn and skip them in the round robin selection algoritm. This will also cause the chain to slow down since round timeouts are computed with an exponential back-off algorithm.

Severity is low since at this stage the committee is trusted.

Recommendations:

We recommend discarding duplicate `NewView` messages

Plasma: Resolved with PR-1388.

Zenith: Verified.

[L-7] Fatal termination of `SyncManager` on stream closure

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

Target

- crates/networking/src/domain/sync_manager.rs#L260-L300

Description:

The peer-to-peer block streaming pipeline (`SyncManager` → `Catchup` → `BlockProcessor`) interprets internal transport or channel closures as fatal events.

When the acceptor for incoming streams closes, the system cancels a shared cancellation token, causing cascading termination across networking, catchup, and block processing.

This behavior can be triggered by Sybil peers, where malicious peers repeatedly open and close streams, saturating stream limits and stressing the `libp2p` accept queue. It can also arise from benign conditions such as transient connectivity loss, resource exhaustion, or peer resets.

In all cases, once `incoming_streams.next()` yields `None`, the current logic halts the `SyncManager` loop, leading `Catchup` and `BlockProcessor` to shut down as well.

The impact on a validator is: block serving stops, catchup is aborted, and the node falls behind the network. This results in missed proposals and votes, prevents participation in quorum certificate formation, and can ultimately render a validator non-functional until manual intervention, despite the system otherwise being healthy.

Recommendations:

We recommend refactoring stream management so that closure of an individual acceptor or channel is handled as a recoverable condition rather than a fatal one. The `SyncManager` should be resilient to `None` events from `incoming_streams`, either by restarting the acceptor, retrying after backoff, or isolating the failure without propagating cancellation across the entire node.

Plasma: Resolved with PR-1387.

Zenith: Verified.

[L-8] Length field width mismatch causes spurious empty frames in block decoding

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

Target

- crates/networking/src/domain/messages/block.rs#L10-L33

Description:

A framing inconsistency exists between how blocks are encoded and how they are decoded on the network. Each `NetworkRawBlock` is serialized with an 8-byte length prefix (`BufMut::put_u64(len)`) followed by the payload.

However, both `inbound_stream_handler.rs` and `outbound_sink_handler.rs` construct `LengthDelimitedCodec::new()`, which defaults to a 4-byte (`u32`) length field.

As a result, when the decoder reads an 8-byte prefix, it interprets the upper 4 bytes as a standalone length. For typical payloads under 4 GB, those upper bytes are zero, producing a spurious empty frame of length 0.

The decoder then consumes the lower 4 bytes as the actual frame length, yielding the real block. This causes every transmitted block to be received as two frames: an empty frame (`NetworkRawBlock(vec![])`), and the real block payload.

Consequently, downstream consumers receive zero-length blocks interleaved with valid ones leading to inefficiencies.

Recommendations:

We recommend aligning encoding and decoding to a consistent framing scheme:

- Configure `LengthDelimitedCodec` to use an 8-byte length field with the intended maximum frame size (20 MB).
- Alternatively, change the sender to emit 4-byte length prefixes if compatibility with default codec settings is desired.

Ensuring both ends agree on the prefix width restores one-to-one block framing and eliminates the empty frame artifacts.

Plasma: Resolved with PR-1294.

Zenith: Verified.

[L-9] Insecure file permissions on Windows enable identity key disclosure

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

Target

- crates/networking/src/adapters/identity_adapter.rs#L143-L148

Description:

On non-Unix platforms, the identity file is written without setting restrictive permissions, the Unix path uses `600`, but the Windows path uses a plain `fs::write`.

Depending on default NTFS ACL inheritance, this may leave the file world-readable (e.g., accessible to `Users` or `Everyone`). A malicious actor with local read access can exfiltrate the private key, impersonate the node's `PeerID` and perform man-in-the-middle of secure channel handshakes.

Recommendations:

We recommend writing the identity file on Windows with an explicit, restrictive DACL and disabled inheritance, granting full control only to the current user (and, optionally, `SYSTEM`).

Plasma: The client acknowledges the issue, stating that Plasma does not support Windows hosts.

[L-10] Insecure transport schemes allow JWT disclosure and engine impersonation

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

Target

- crates/execution_proxy/src/transport/authenticated.rs#L37-L54

Description:

The Engine API client accepts http, https, ws, wss, and file without host restrictions.

During connection, `InnerTransport::connect` routes http/https to `connect_http` and ws/wss to `connect_ws`, and always attaches an `Authorization: Bearer <JWT>` header/token. TLS is only enabled for https (and by implication wss); when http or ws is used, the JWT is sent in cleartext.

Because there is no restriction to loopback, configuring `engine_api_url` to a remote `http://` or `ws://` endpoint exposes the JWT to on-path interception, enabling a malicious actor to impersonate the engine and issue arbitrary privileged calls.

Recommendations:

We recommend enforcing authenticated, encrypted transport and scoping insecure schemes to localhost only by disallowing http/ws for non-local hosts (or disable entirely).

Plasma: The client acknowledges the issue stating that the risk is acceptable.

[L-11] Multi-chunk block catch-up stalls after first chunk

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

Target

- crates/node/src/domain/catchup_state.rs#L59-L62

Description:

The node's block catch-up process is designed to handle large synchronization ranges by breaking them into smaller, manageable chunks. This is managed by the `CatchupPhaseTransitioner`.

In `handle_range_request`, if a block range request is received that is larger than `max_catchup_range_size` (by default, `10_000`) while the `CatchupPhase` is `Idle`, the system correctly trims the request to the first chunk. However, it fails to record the original, larger target height by calling `update_pending_to`.

After successfully processing this first chunk, the `handle_catchup_completed` function is called. It checks for any pending catch-up ranges to continue the synchronization process. Because the original target height was never stored, the system incorrectly assumes the entire catch-up is finished. It then transitions back to the `Idle` state, and no further block chunks are requested.

This results in the node's synchronization stalling, requiring the node to receive another unknown block to resume the catchup, preventing smooth sync-up with the network.

Recommendations:

Consider ensuring that when a catch-up range is trimmed during the `Idle` phase, the `pending_to` state is updated with the original target block height. This will allow `handle_catchup_completed` to correctly schedule subsequent chunks until the node is fully synchronized.

Recommendations:

Plasma: Resolved with PR-1291.

Zenith: Verified.

4.3 Informational

A total of 9 informational findings were identified.

[I-1] Mismatched block size limits

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

Target

- crates/types/src/domain/consensus/constants.rs#L6

Description:

The networking protocol enforces inconsistent maximum size limits for raw blocks between the sending and receiving ends of a stream. On the sender's side, the `encode` function for `NetworkRawBlock` checks if a block's size exceeds `MAX_NETWORK_RAW_BLOCK_SIZE`, which is defined as 20 MB.

However, the receiver, within the `build_inbound_stream` function in `crates/networking/src/utils/inbound_stream_handler.rs`, utilizes `tokio_util::codec::LengthDelimitedCodec::new()`. This constructor applies a default maximum frame length of 8 MB, which is significantly lower than the sender's 20 MB limit.

As there is no actual impact besides the inconsistency, we classify this issue as informational.

Recommendations:

Consider synchronizing the maximum frame size limits on both the sending and receiving sides of a network stream. The `LengthDelimitedCodec` on the receiving end should be explicitly configured with the same `MAX_NETWORK_RAW_BLOCK_SIZE` value used by the sender.

Plasma: Resolved with PR-1271.

Zenith: Verified.

[I-2] Trusted peers bypass connection limits without safeguards enabling denial of service

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

Target

- crates/networking/src/domain/network_manager.rs#L255-L265

Description:

The networking layer currently allows trusted peers (explicitly configured or allowed peers) to bypass global connection limits.

While this ensures connectivity with critical peers, it introduces a denial-of-service (DoS) risk. If a trusted peer becomes faulty or is compromised by a malicious actor, it can consume all available inbound or outbound slots.

Since the bypass mechanism exempts trusted peers from enforcement, this can result in connection exhaustion, degraded availability, and potential network partitioning. In adversarial scenarios, an attacker could deliberately exploit this trust exemption to destabilize consensus.

Recommendations:

We recommend retaining the bypass mechanism for trusted peers but introducing per-trusted-peer caps. enforce a maximum number of simultaneous connections per trusted peer to prevent monopolization of network slots.

Plasma: The client acknowledges the issue, stating that the risk is acceptable.

[I-3] Unused `enable_peer_scoring` flag

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

Target

- crates/node/src/domain/config.rs#L173

Description:

The configuration parameter `enable_peer_scoring` is defined in the networking layer but is never consumed in the runtime logic.

As a result, peer scoring, which should penalize misbehaving or unresponsive peers, is not enforced.

Recommendations:

We recommend integrating the `enable_peer_scoring` flag into the peer management subsystem or removing this flag if it is not needed.

Plasma: The client acknowledges the issue stating that the peer scoring subsystem was removed so this flag is vestigial and not meant to be used. They will remove it in the future.

[I-4] Incorrect error variant returned for oversized block envelopes

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

Target

- crates/networking/src/domain/messages/envelope.rs#L19-L25

Description:

When creating a `RawBlockEnvelope`, the constructor incorrectly returns the `VoteEnvelopeTooLarge` error variant if the provided data exceeds `MAX_BLOCK_ENVELOPE_SIZE`.

This misclassification makes it difficult to distinguish between oversized vote envelopes and oversized block envelopes, reducing error clarity and potentially misleading operators or automated error handling.

Recommendations:

We recommend updating the constructor of `RawBlockEnvelope` to return the correct `BlockEnvelopeTooLarge` error variant when the block size exceeds its limit.

Plasma: Resolved with PR-1401.

Zenith: Verified.

[I-5] Outdated comment for `MAX_NETWORK_RAW_BLOCK_SIZE` consensus constant

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

Target

- crates/types/src/domain/consensus/constants.rs#L6

Description:

The constant `MAX_NETWORK_RAW_BLOCK_SIZE` is defined as 20MB (`20 * 1024 * 1024`). However, the accompanying comment incorrectly states the value is _12MB_. This outdated comment can be misleading to developers, potentially causing them to misinterpret the maximum raw block size limit defined in the constant.

Recommendations:

Consider updating the comment to reflect the actual value of 20MB to avoid confusion.

Plasma: Resolved with PR-1372.

Zenith: Verified.

[I-6] Unchecked `UnicastRequest` data length in `UnicastRequest::into_request_rpc`

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

Target

- crates/networking/src/domain/unicast.rs#L75-L76

Description:

The `UnicastRequest::into_request_rpc()` function uses `RawVoteEnvelope::from()` and `RawNewViewEnvelope::from()` to convert the byte payload (`self.data`) into `RequestRpc::Vote` and `RequestRpc::NewView` variants.

These `From` implementations do not validate the length of the provided data. Consequently, incoming `Vote` and `NewView` messages can exceed the intended maximum size limits, potentially leading to resource exhaustion or other denial-of-service vulnerabilities. Other message types, such as `BlockProposal`, correctly use a `::new()` constructor which enforces the `MAX_BLOCK_ENVELOPE_SIZE` constraint.

Recommendations:

Consider replacing the use of `::from()` with constructors that enforce size limits, such as `RawVoteEnvelope::new()` and `RawNewViewEnvelope::new()`, to ensure that all unicast messages are validated adequately against size constraints.

Plasma: Resolved with PR-1400.

Zenith: Verified.

[I-7] Unvalidated QC `proposer_index` during block validation

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

Target

- crates/consensus/src/validation/validator_block.rs#L462-L497

Description:

When validating a new block in `validate_block_internal` in `crates/consensus/src/validation/validator_block.rs`, the attached Quorum Certificate (QC) is checked via the `validate_block_qc` function. This function correctly verifies that the QC contains a valid quorum of signatures from the committee.

However, the `proposer_index` field within the QC is not validated. This field is intended to identify the validator that proposed the block to which the QC refers. As a result, a block with a valid QC that specifies an arbitrary `proposer_index` can be built and propagated in the network, which will be accepted and added as part of the chain.

Currently, this field does not appear to be used for any critical logic, so the immediate impact is low. However, this represents a latent vulnerability. Future code changes may assume that the `proposer_index` is a validated and trustworthy value. Relying on this unvalidated data could introduce difficult-to-detect bugs or security vulnerabilities.

Recommendations:

Consider enhancing the QC validation logic in `validate_block_qc` to verify the `proposer_index`.

Plasma: Resolved with PR-1397.

Zenith: Verified.

[I-8] Unbounded timeout backoff exponent enables panic and denial of service

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

Target

- crates/consensus/src/state_transition/state_transition_service.rs#L64-L89

Description:

After each event, the service computes a timeout reset with an exponent derived from the "view gap" `view_diff = post_view.saturating_sub(post_qc_view).saturating_sub(1)` and emits `CoreEffect::ResetTimeout(view_diff)`. The timer then evaluates `current_duration = base * factor.powi(view_diff as i32)`.

During `NewView` aggregation, `post_view` can jump to `aggregated_view + 1` while `post_qc_view` is the highest QC inside that aggregation. Because protocol rules only require `QC.view < NewView.view`, the difference `post_view - post_qc_view` can be unbounded.

Although the reset path clamps the integer to `i32::MAX`, evaluating `factor.powi(2_147_483_647)` can yield `inf` and trigger a panic in `Duration::mul_f64`, or, if finite but enormous, produce an effectively infinite sleep.

While recovery mode forces a base timeout when the network was not operational, a single large aggregation once operational can still explode the exponent and destabilize the node.

Scenario:

- The node is operational (so the code uses the "difference" path).
- It aggregates a threshold of `NewView` messages for a very large view `V` while the highest QC among them is much smaller `H` (still valid and present locally).
- After processing, `post_view ≈ V + 2` and `post_qc_view = H`. Then `view_diff = (V + 2 - H) - 1 = V + 1 - H`. If `V – H` is large (e.g., thousands or millions), the exponent is huge and the timer reset computes `factor.powi()`.

Recommendations:

We recommend introducing strict bounds to prevent exponential blowups. The exponent should be clamped to a small constant (such as 16—32) before invoking powi, and the

resulting duration should also be capped at a reasonable upper limit to ensure the timer never sleeps indefinitely.

Plasma: Resolved with PR-1373.

Zenith: Verified.

[I-9] Network manager continuously attempts to reconnect blacklisted peers

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

Target

- crates/networking/src/domain/network_manager.rs#L267-L279

Description:

The `NetworkManager` periodically attempts to reconnect to disconnected peers. This is handled by the `handle_reconnect_tick` function, which calls `connection_manager.peers_to_reconnect()` to get a list of peers to dial.

Peers that send invalid data, for example, a gossip message that fails to be decoded, are blacklisted via `Swarm::blacklist()`. This action correctly blocks the peer at the libp2p swarm level, preventing further connections from them.

However, the `ConnectionManager` is not made aware of this blacklisting. Therefore, it continues to consider the blacklisted peer a candidate for reconnection. On each `handle_reconnect_tick` execution, the `NetworkManager` will attempt to dial the blacklisted peer, which will immediately fail. This results in a continuous loop of failed connection attempts, leading to wasted resources and unnecessary error logging.

Recommendations:

Consider informing the `ConnectionManager` about blacklisted peers to exclude them from the reconnection attempts

Plasma: Acknowledged.