ClarityAlliance

NAKAMOTO UPGRADE SECURITY REVIEW

Conducted by:
KRISTIAN APOSTOLOV, ABA, MARCHEV, ARABADZHIEV

SEPTEMBER 20TH, 2024

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

1. About Clarity Alliance

Clarity Alliance is a team of expert whitehat hackers specialising in securing protocols on Stacks.

They have disclosed vulnerabilities that have saved millions in live TVL and conducted thorough reviews for some of the largest projects across the Stacks ecosystem.

Learn more about Clarity Alliance at clarityalliance.org.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

2. Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Clarity Alliance to perform a security assessment.

This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Clarity Alliance’s position is that each company and individual are responsible for their own due diligence and continuous security. Clarity Alliance’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree
to analyze.

The assessment services provided by Clarity Alliance are subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis.

Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third parties. Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Clarity Alliance does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

3. Introduction

A time-boxed security review of the Nakamoto upgrade implementation, where Clarity Alliance reviewed the scope, whilst simultaneously building out a testing suite for the protocol.

4. About Stacks Nakamoto Upgrade

The Nakamoto Release is an upcoming hard fork on the Stacks network designed to bring several benefits, chief among them are increased transaction throughput and 100% Bitcoin finality. Learn about the Nakamoto Activation sequence here: Nakamoto Activation Sequence

Bitcoin block 840,360 marked the start of the multi-phase Nakamoto mainnet rollout.

With Nakamoto, Stacks block production would no longer be tied to miner elections. Instead, miners produce blocks at a fixed cadence, and the set of PoX Stackers rely on the miner elections to determine when the current miner should stop producing blocks and a new miner should start. This blockchain will only fork if 70% of Stackers approve the fork, and chain reorganization will be as difficult as reorganizing Bitcoin.

The Nakamoto release brings many new capabilities and improvements to the Stacks blockchain by focusing on a set of core advancements: improving transaction speed, enhancing finality guarantees for transactions, mitigating Bitcoin miner MEV (miner extractable value) opportunities that affect PoX, and boosting robustness against chain reorganizations.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

5.1 Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2 Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

CONTENTS

6. Security Assessment Summary

The engagement scope was localized on Nakamoto Upgrade critical changes, introduced to the `stacks-core` codebase, contained within the following PRs:

PR 4728 fix: `add-signer-key` to synthetic `stack-aggregation-increase` event

PR 4781 modify nakamoto block header to use `Vec<MessageSignature>`

PR 4788 Add v0 signer `process_event` paths

PR 4807 feat: gather v0 block signatures from stackerdb

PR 4827 Feat/tenure extend transactions

PR 4877 Feat/nakamoto block push

PR 4879 Feat: PoX punish/reward via bitvec

PR 4902 Feat: push blocks to signer set and add `/v3/blocks/upload/`

PR 4919 End of tenure test

PR 4930 MARF'ed Nakamoto chainstate

ClarityAlliance
Security Review

Nakamoto Upgrade

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

7. Executive Summary

Over the course of the security review, Kristian Apostolov, ABA, marchev, Arabadzhiev engaged with Stacks Foundation to review Stacks Nakamoto Upgrade. In this period of time a total of **34** issues were uncovered.

Protocol Summary

Protocol Name	Stacks Nakamoto Upgrade
Repository	https://github.com/stacks-network/stacks-core
Date	September 20th, 2024
Protocol Type	DLT Upgrade

Findings Count

Severity	Amount
Medium	3
Low	5
QA	26
Total Findings	**34**

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

Summary of Findings

ID	Title	Severity	Status
[M-01]	Block Bitvec Header Incorrectly Validated Against Miner Block Commit Punishments	Medium	Resolved
[M-02]	Insufficient Error Handling When Calling insert_burn_block	Medium	Resolved
[M-03]	Sortition DB Instantiation Creates Wrong Schema Version	Medium	Resolved
[L-01]	Missing Error Handling in Case of Failed Block Push	Low	Resolved
[L-02]	Failing to Retrieve Nakamoto Staging Blocks Version Will Wipe Entire DB	Low	Resolved
[L-03]	Block Timestamp Can Be 15 Seconds into the Future	Low	Resolved
[L-04]	Miner signature hash does not contain POX treatment bitvec	Low	Resolved
[L-05]	Incorrect Tenure Chainstate Schema Migration from Version 3 to Version 4	Low	Resolved
[QA-01]	The time complexity of NakamotoChainState::check_pox_bitvector could be reduced from O(mn) to O(n)	QA	Resolved
[QA-02]	Block PoX Bitvec Header Lacks Edge Validations Against Miner Block Commit Punishments	QA	Acknowledged
[QA-03]	Typographical Errors	QA	Acknowledged
[QA-04]	Misleading Warning Message When Submitting Proposal Response to .signers Fails	QA	Acknowledged
[QA-05]	Inconsistent RESTful URI Design in RPCPostBlockRequestHandler	QA	Acknowledged
[QA-06]	Misleading rustdoc for SignerRunLoop#main_loop	QA	Acknowledged
[QA-07]	Missing Implementation for Signer#updateSigner()	QA	Acknowledged
[QA-08]	The StackerDB#send_message_bytes_with_retry() implementation could be simplified	QA	Acknowledged

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

Summary of Findings

ID	Title	Severity	Status
[QA-09]	BACKOFF_MAX_INTERVAL should not exceed BACKOFF_MAX_ELAPSED	QA	Acknowledged
[QA-10]	Improve new_tenure and tenure_extended Variable Naming	QA	Acknowledged
[QA-11]	Sign Coordinator v0 Logging Discrepancies	QA	Acknowledged
[QA-12]	Wrapping versus Saturating Reward Cycle Inconsistencies	QA	Acknowledged
[QA-13]	Move force_send Configuration to ConnectionOptions	QA	Acknowledged
[QA-14]	Improve NET Relayer Logging	QA	Acknowledged
[QA-15]	NakamotoBlocksData Consensus Deserialization Can Be Optimized	QA	Acknowledged
[QA-16]	Use Constants Instead of Magic Numbers in RPCRequestHandler	QA	Acknowledged
[QA-17]	Improve Logging in Nakamoto Chainstate Module Code	QA	Acknowledged
[QA-18]	NakamotoStagingBlocksConnRef::has_children is never used	QA	Acknowledged
[QA-19]	Indistinguishable NakamotoBlockBuilder::load_tenure_info Error Messages	QA	Resolved
[QA-20]	mod::NakamotoChainState::check_pox_bitvector can be simplified	QA	Acknowledged
[QA-21]	Use Descriptive Variable Names	QA	Acknowledged
[QA-22]	Unused Imports	QA	Partially Resolved
[QA-23]	Inconsistent Ordering of Match Cases with StacksMessageType Throughout the Code	QA	Acknowledged
[QA-24]	Incorrect Sortition DB Schema 4 SQL Commands	QA	Resolved
[QA-25]	Improve Nakamoto Node Miner Thread Debug Logging	QA	Resolved
[QA-26]	Continue Tenure Directive Logging Ambiguities	QA	Acknowledged

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

8. Findings

8.1. Medium Findings

[M-01] Block Bitvec Header Incorrectly Validated Against Miner Block Commit Punishments

PR: 4879

Description

In the `NakamotoChainState` component, the PoX bitvec is validated via the `check_pox_bitvector` function. This function ensures that the current block's bitvec header matches the miner's block commit punishments bitvec according to specific rules.

To perform validations, a bit array, `bitvec_values`, is created. This array associates each address from the miner's block commit punishment array to a reward set entry and then correlates it with the block's bitvec header.

There are three main validations to be performed on this array:

```
// if any of them are 0, punishment is okay.
// if all of them are 1, punishment is not okay.
// if all of them are 0, *must* have punished
```

An issue arises specifically when the block header's bitvec is smaller than the reward set. In this case, since the default value used when populating the intermediary `bitvac_values` vector is true (1), the case if *all of them are 0, must have punished* is not enforced.

The validation is never reached because, when checking that the `bitvac_values` are 0 (via the `all_0` variable):

```
// if all of them are 0, *must* have punished
    let bitvec_values: Result<Vec<_>, ChainstateError> = // ... code ...

    let all_0 = bitvec_values.iter().all(|x| !x);
    if all_1 {
      // ... code ...
    } else if all_0 {
        if treated_addr.is_reward() {
          warn!(
                                    "Invalid Nakamoto block: rewarded PoX address when bitvec containe
              "reward_address" => %treated_addr.deref(),
              "bitvec_values" => ?bitvec_values,
          );
          return Err(ChainstateError::InvalidStacksBlock(
          "Bitvec does not match the block commit's PoX handling".into(),
          ));
      }
    }
```

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

Description

The `else` branch will not execute, as the extra padded values are `1`, thus allowing the case where all `bitvec_values` are 0, but the treated address is rewarded.

Therefore, a situation can occur where the block header's bitvec is smaller than the reward set, and the miner's block punishment is a reward, but the current block's bitvec indicates a punishment. In this case, the check is skipped, and no error is returned.

Recommendation

Create the `bitvec_values` up to the length of the `active_reward_set`'s addresses enumeration.

[M-02] Insufficient Error Handling When Calling `insert_burn_block`

PR: develop

Description

There is a discrepancy in the way failed calls to `insert_block` and `insert_burn_block` are handled. When the `insert_block` function fails, a custom panic is triggered:

signer.rs#L403-L405

```
self.signer_db
    .insert_block(&block_info)
    .unwrap_or_else(|_| panic!("{self}: Failed to insert block in DB"));
```

However, for the `insert_burn_block` function, a failure only results in a custom message being logged at the `warn` level:

signer.rs#L403-L405

```
if let Err(e) = self.signer_db
    .insert_burn_block(burn_header_hash, *burn_height, received_time)
{
    warn!(
        "Failed to write burn block event to signerdb";
        "err" => ?e,
        "burn_header_hash" => %burn_header_hash,
        "burn_height" => burn_height
    );
}
```

Since both functions are used at least once in the execution paths of process_event, this means that when `process_event` is called, unexpected situations can occur. This is because normal STX blocks and burnchain blocks are not treated equally when an error occurs during their persistence to the database.

Recommendation

Add error handling logic that panics or throws a custom error whenever a call to the `insert_burn_block` function fails.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[M-03] Sortition DB Instantiation Creates Wrong Schema Version

PR: 4879

Description

When instantiating the `SortitionDB`, the schema version ends up being incorrect due to the order in which migrations are applied. Specifically, the schema 8 migration is applied after the schema 9 migration (the latest), resulting in the final sortition schema version field being set to 8 (stale).

The issue occurs because `SortitionDB::apply_schema_9` is executed before the `SortitionDB::apply_schema_8_migration` function call.

```
SortitionDB::apply_schema_8_tables(&db_tx, epochs_ref)?;
SortitionDB::apply_schema_9(&db_tx, epochs_ref)?;
// ... code ...
db_tx.commit()?;
// NOTE: we don't need to provide a migrator here because we're not migrating
self.apply_schema_8_migration(None)?;
```

`apply_schema_9` sets the schema version to 9:

```
tx.execute(
    "INSERT OR REPLACE INTO db_config (version) VALUES (?1)",
    &["9"],
)?;
```

However, `apply_schema_8_migration` then sets it to 8:

```
tx.execute(
    "INSERT OR REPLACE INTO db_config (version) VALUES (?1)",
    &["8"],
)?;
```

Although the internal fields of the database would be correct, the schema version itself would be wrong. This discrepancy could lead to potential database corruption, as some parts of the codebase behave differently with schema 8 versus schema 9.

Recommendation

Move the schema 9 (and all subsequent schema improvements) after the `apply_schema_8_migration` call.

ClarityAlliance
Security Review

Nakamoto Upgrade

8.2. Low Findings

[L-01] Missing Error Handling in Case of Failed Block Push

PR: 4879

Description

When processing a `SignerMessage::BlockPushed` event, the `Signer` fails to handle any `ClientError`s that might occur when trying to post a Nakamoto block to a node:

```
SignerMessage::BlockPushed(b) => {
    let block_push_result = stacks_client.post_block(b);
    info!(
        "{self}: Got block pushed message";
        "block_id" => %b.block_id(),
        "signer_sighash" => %b.header.signer_signature_hash(),
        "push_result" => ?block_push_result,
    );
}
```

The implementation only handles the happy path scenario, and the application does notreact to errors in any way.

Recommendation

Handle any `ClientError`s returned by `stacks_client.post_block(b)` for example, by logging a warning via `warn!()` or handling it more appropriately depending on the severity and expectation of such errors.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[L-02] Failing to Retrieve Nakamoto Staging Blocks Version Will Wipe Entire DB

PR: 4930

Description

When the Stacks core reads the internal SQL database containing the Nakamoto staging database, the code will always attempt to migrate the schema if the current version is less than 2.

```
pub fn open_nakamoto_staging_blocks(
    path: &str,
    readwrite: bool,
) -> Result<NakamotoStagingBlocksConn, ChainstateError> {
    let exists = fs::metadata(&path).is_ok();
    // ... code ...
    if !exists {
        // ... code ...
    } else if readwrite {
    Self::migrate_nakamoto_staging_blocks(&conn)?;
    }
    // ... code ...
}
pub fn migrate_nakamoto_staging_blocks(conn: &Connection) -> Result<
(), ChainstateError> {
    let mut version = Self::get_nakamoto_staging_blocks_db_version(conn)?;
    if version < 2 {
       debug!("Migrate Nakamoto staging blocks DB to schema 2");
       for cmd in NAKAMOTO_STAGING_DB_SCHEMA_2.iter() {
           conn.execute(cmd, NO_PARAMS)?;
       }
    // ... code ...
}
```

However, when retrieving the DB version via the `get_nakamoto_staging_blocks_db_version` function, if a read error occurs, it defaults to version 1 instead of reverting:

```
Err(e) => {
    debug!("Failed to get Nakamoto staging blocks DB version: {:?}", &e);
    return Ok(1);
}
```

Defaulting to schema 1 will then apply schema 2, which will first drop the existing table before recreating the database:

```
pub const NAKAMOTO_STAGING_DB_SCHEMA_2: &'static [&'static str] = &[
    r#"
    DROP TABLE nakamoto_staging_blocks;
    "#,
```

This behavior is dangerous because, in the current configuration, if there are any issues (e.g., storage/IO) with reading the schema version, the entire database will be dropped.

Recommendation

Either modify the `NAKAMOTO_STAGING_DB_SCHEMA_2` schema to simply update the existing table in case the mentioned issue occurs, or revert when failing to determine the current database schema, and do not default to schema 1.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[L-03] Block Timestamp Can Be 15 Seconds into the Future

PR: 4930

Description

The newly added block timestamp is described as follows:

```
/// A Unix time timestamp of when this block was mined, according to the
    // miner.
    /// For the signers to consider a block valid, this timestamp must be:
    /// * Greater than the timestamp of its parent block
    /// * Less than 15 seconds into the future
    pub timestamp: u64,
```

However, during validation:

```
if self.block.header.timestamp > get_epoch_time_secs() + 15 {
      return Err(BlockValidateRejectReason {
      reason_code: ValidateRejectCode::InvalidBlock,
      reason: "Block timestamp is too far into the future".into(),
      });
}
```

the second requirement is not fully respected because the timestamp can be exactly 15 seconds into the future, making the actual enforced condition *less than 16 seconds into the future*, which is incorrect.

Recommendation

Change the block header timestamp future check from `>` to `>=` so that it does not consider 15 seconds into the future as valid.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[L-04] Miner signature hash does not contain POX treatment bitvec

PR: 4930

Description

To calculate the hash a miner must sign, the `struct::NakamotoBlockHeader::miner_signature_hash` function is called.

This resulting hash is calculated on all fields from the `NakamotoBlockHeader` structure except the signatures, the signer signature, and, of course, the miner signature itself since it has not been signed yet.

```
/// Calculate the message digest for miners to sign.
/// This includes all fields _except_ the signatures.
pub fn miner_signature_hash(&self) -> Sha512Trunc256Sum {
    self.miner_signature_hash_inner()
        .expect("BUG: failed to calculate miner signature hash")
}
```

The inner `miner_signature_hash-inner` function, however, does not take into consideration the newly added BitVec field.

From a consensus point of view, this issue is not relevant as the `pox_treatment` is taken into consideration by both the signer's signed message digest and when the `NakamotoBlockHeader` structure is serialized for consensus via the `consensus_serialize` function.

The impact is a deviation from the aforementioned documented intent.

Recommendation

Add the `pox_treatment` field when calculating the hash in the `miner_signature_hash-inner` function or modify the comment to indicate the current, different but intended behavior.

[L-05] Incorrect Tenure Chainstate Schema Migration from Version 3 to Version 4

PR: 4827

Description

Due to a typo in the SQL command when migrating the transaction chainstate schema from version 3 to version 4, the operation fails.

Detailed Description

When a schema migration is initiated via the `stackslib\\src\\chainstate\\stacks\\db\\mod.rs::instantiate_db` function, the `StacksChainState::apply_schema_migrations` function is called:

```
StacksChainState::apply_schema_migrations(&tx, mainnet, chain_id)?;
```

During the migration from chainstate schema version 3 to 4, the `NAKAMOTO_CHAINSTATE_SCHEMA_1` commands are executed:

```
"3" => {
    // migrate to nakamoto 1
    info!("Migrating chainstate schema from version 3 to 4: nakamoto support");
    for cmd in NAKAMOTO_CHAINSTATE_SCHEMA_1.iter() {
        tx.execute_batch(cmd)?;
    }
}
```

The `stackslib/src/chainstate/nakamoto/mod.rs::NAKAMOTO_CHAINSTATE_SCHEMA_1` commands also include the stackslib\src\chainstate\nakamoto\tenure.rs::NAKAMOTO_TENURES_SCHEMA_1 sub-commands. It is in this schema that, on line 134, a typo in the SQL command leads to migration failure:

`cause INETGER NOT NULL,` should be `cause INTEGER NOT NULL,`

Recommendation

Resolve the SQL command typo.

8.3. QA Findings

[QA-01] The time complexity of NakamotoChainState::check_pox_bitvector could be reduced from O(mn) to O(n)

PR: 4879

Description

The function `check_pox_bitvector()` currently operates with a time complexity of $O(mn)$, where m is the number of treated addresses and n is the number of rewarded addresses. This is because, for each treated address, the function iterates over the entire `rewarded_addresses` list, which can lead to inefficiencies as the size of the data grows.

Recommendation

To reduce the time complexity from $O(mn)$ to $O(n)$, we recommend using a collection like `HashMap` or `HashSet` to pre-process `rewarded_addresses` . By storing the `rewarded_addresses` in a `HashSet` , lookups for each treated address can be performed in constant time of $O(1)$, which would reduce the overall complexity of the function to $O(n)$. This will significantly improve performance, especially when dealing with large datasets.

[QA-02] Block PoX Bitvec Header Lacks Edge Validations Against Miner Block Commit Punishments

PR: 4879

Description

In the `NakamotoChainState` component, the PoX bitvector is validated via the `check_pox_bitvector` function. This function ensures that the current block's bitvec header matches the miner's block commit punishments bitvector.

However, there are two corner cases where no validation is performed.

The first case occurs when the miner's block commit punishments address list is empty. In this scenario, the `if` statement that performs the validations is skipped:

```
if !tenure_block_commit.treatment.is_empty()
```

This means that if the miner's block commit punishments (`tenure_block_commit.treatment` variable) are empty, the current block's bitvec header (`tenure_block_pox.treatment`) can be any value, as it is not validated against the miner's block punishments, as per the mentioned requirement.

The second case occurs when the miner's punishments list contains burn addresses. In this scenario, no check is performed:

```
if treated_addr.is_burn() {
      // Don't need to assert anything about burn addresses.
      // If they were in the reward set, "punishing" them is meaningless.
      continue;
}
```

Recommendation

Document these two special cases and implement validation for both of them.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-03] Typographical Errors

PR: develop

Description

Throughout the codebase, there are several typos. Some are in more sensitive locations and may cause other issues, which were noted in separate findings, while others are in less critical places such as comments or code naming.

Of particular importance is the `transfering -> transferring` typo, which is also present in the SQL column naming. As such, a mass fix should not be done, but rather on a case-by-case basis.

The following is a list of typos found in the codebase that appear in several locations and can be replaced directly if searched as whole words:

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

- desierialized → deserialized
- contruct → construct
- DkgPublicshares → DkgPublicShares
- signaure → signature
- atomicbool → atomic bool
- RPCERror → RPCError
- respues → request
- Nakamato → Nakamoto
- communciation → communication
- StacksAdress → StacksAddress
- addands → addends
- reamining → remaining
- structions → structs
- suprising → surprising
- Consennsus → Consensus
- serializationn → serialization
- contibuted → contributed
- hext → text
- unrecovarable → unrecoverable
- oudated → outdated
- Retreieve → Retrieve
- decryoted → decrypted
- cyrptographic → cryptographic
- hexademical → hexadecimal
- unspecificed → unspecified
- sortitin → sortition
- signtaure → signature
- dispatcheer → dispatcher
- addresess → addresses
- constituant → constituent
- minimial → minimal
- recognizeable → recognizable
- BittcoinTxInputStructured → BitcoinTxInputStructured
- accomodate → accommodate
- heigth → height
- discontiguous → discontinuous
- committedto → committed to
- Unparseable → Unparsable
- parseable → parsable
- occured → occurred
- unsoliciated → unsolicited
- CoordiantorError → CoordinatorError
- get_prepare_phase_end_sortition_id_for_reward_ccyle →
- get_prepare_phase_end_sortition_id_for_reward_cycle
- reawrd → reward
- sorition → sortition
- conver → convert
- superceded → superseded
- interpreteted → interpreted

CONTENTS

- burchain → burnchain
- check_intneded_sortition → check_intended_sortition
- DescendencyStubbedSortitionHandle → DescendenceStubbedSortitionHandle
- infallable → infallible
- sentinal → sentinel
- attched → attached
- sortiiton → sortition
- cosnistent → consistent
- reorog → reorg
- procesed → processed
- sortitoin → sortition
- Nakamaoto → Nakamoto
- sortiton → sortition
- parnet → parent
- begain → began
- epcoh2 → epoch2
- becuase → because
- Nakamto → Nakamoto
- slots_occuppied → slots_occupied
- recieve → receive
- InvalidChildOfNakomotoBlock → InvalidChildOfNakamotoBlock
- proessed → processed
- miroblock → microblock
- non-sensical → nonsensical
- unconfiremd → unconfirmed
- confiremd → confirmed
- issueing → issuing
- backpr → backptr
- descendents → descendants
- refered → referred
- anestor → ancestor
- naonseconds → nanoseconds
- smae → same
- exepcted → expected
- uncommmitted → uncommitted
- expanaded → expanded
- unkonwn → unknown
- incomaptible → incompatible
- tranasction → transaction
- accouting → accounting
- offerred → offered
- incrementially → incrementally
- aprse → parse
- appned → append
- sorition_id → sortition_id
- begining → beginning
- issuring → issuing
- retun → return

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

- snippit → snippet
- tesnet → testnet
- Cacluated → Calculated
- contibute → contribute
- contrac → contract
- succesful → successful
- Blook → Block
- bytess → bytes
- agains → against
- sucess → success
- sorittion → sortition
- wnated_tenures → wanted_tenures
- clobberring → clobbering
- instantiang → instancing
- NakamotoDwonloadStateMachine → NakamotoDownloadStateMachine
- mantained → maintained
- akamotoDownloaderStateMachine → NakamotoDownloaderStateMachine
- reqeust → request
- NakamotTenureDownloader → NakamotoTenureDownloader
- tenurein → tenure
- advanceement → advancement
- perserve → preserve
- blcok → block
- HttpReqeust → HttpRequest
- Inavlid → Invalid
- bufferring → buffering
- consturctor → constructor
- Connction → Connection
- mroe → more
- enoding → ending
- loewst → lowest
- target_reward_cyle → target_reward_cycle
- scaning → scanning
- epcoh2x → epoch2x
- epcoh → epoch
- enusre → ensure
- garabage → garbage
- ommitted → omitted
- minimim → minimum
- netwrk → network
- allwed → allowed
- neighors → neighbors
- desigend → designed
- exisitng → existing
- exceeeds → exceeds
- incuring → incurring
- neigbors → neighbors

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

- reesolution → resolution
- converation → conversation
- messsage → message
- maximially → maximally
- existant → existent
- malforemd → malformed
- thestart → the start
- inventroy → inventory
- witih → with
- rerog → reorg
- recieved → received
- local_adddr → local_addr
- intution → intuition
- messaege → message
- deliberatly → deliberately
- conencted → connected
- converation → conversation
- failrue → failure
- Aleady → Already
- forwraded → forwarded
- previosuly → previously
- beyind → beyond
- succesfully → successfully
- Burnchian → Burnchain
- transacctions → transactions
- stakcs → stacks
- sortititon → sortition
- election_sortiton → election_sortition
- recipent → recipient
- preceeds → precedes
- becase → because
- inplement → implement
- StackerDBChnnel → StackerDBChannel
- Cointer → Counter
- Implmentation → Implementation
- Sychronously → Synchronously
- threeads → threads
- earliersiblings → earlier siblings
- restoroe → restore
- canoincal → canonical
- immutible → immutable
- coordiantor → coordinator
- synchronise → synchronize
- sor/tition → sortition
- actualy → actually
- proces → process

Recommendation

Resolve all indicated typos.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-04] Misleading Warning Message When Submitting Proposal Response to .signers Fails

PR: 4807

Description

If an error occurs when broadcasting a block response to the Stacks node, an incorrect warning message is logged:

```
Err(e) => {
      warn!("{self}: Failed to send block rejection to stacker-db: {e:?}",);
  }
```

At this point, the response could be either `BlockResponse::accepted()` or `BlockResponse::rejected()`. However, the warning message implies that the failed response submission is a rejection, which is misleading.

Recommendation

Update the warning message as follows:

```
crate::monitoring::increment_block_responses_sent(accepted);
            }
            Err(e) => {
-               warn!
- ("{self}: Failed to send block rejection to stacker-db: {e:?}",);
+ warn!
+ ("{self}: Failed to send block response to stacker-db: {e:?}",);
            }
        }
self.signer_db
```

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-05] Inconsistent RESTful URI Design in `RPCPostBlockRequestHandler`

PR: 4902

Description

The `RPCPostBlockRequestHandler` defines the static URI path `/v3/blocks/upload/`, which includes the verb "upload." This design is inconsistent with RESTful principles that emphasize using URIs to represent resources (nouns), while actions (verbs) are conveyed via HTTP methods (such as `GET`, `POST`, `PUT`, `DELETE`). Including verbs in URIs makes the API harder to maintain and reduces clarity by mixing the action with the resource representation.

Additionally, this design is inconsistent with the `getblock_v3.rs` handler, which follows a RESTful URI pattern:

```
GET /v3/blocks/{blockId}
```

Recommendation

To achieve consistency and adhere to RESTful design principles, the URI `/v3/blocks/upload/` should be refactored to remove the verb. For example, change the URI to `/v3/blocks` and rely on the HTTP method (`POST` in this case) to indicate the "upload" action. This would align the design with the existing `getblock_v3.rs`, where the resource (`block_id`) is clearly represented by the URI, and actions are performed based on the HTTP method.

Additionally, consider removing the trailing slash from the URL as this also goes against good RESTful API design practices.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-06] Misleading rustdoc for `SignerRunLoop#main_loop()`

PR: 4788

Description

The rustdoc of the `SignerRunLoop#main_loop()` is misleading. It states:

> `Once it has polled for events, they are fed into run_one_pass(). This continues until either run_one_pass() returns false, or the event receiver hangs up.`

However, the `run_one_pass()` function does not return a `bool` but `Option<R>` instead. The condition that is checked is if `run_one_pass()` returns `Some(final_state)`, in which case the processing stops.

Recommendation

Update the rustdoc as follows:

```
/// This is the main loop body for the signer. It continuously receives
    // events from
    /// `event_recv`, polling for up to `self.get_event_timeout
    //()` units of time. Once it has
    /// polled for events, they are fed into `run_one_pass
    //()`. This continues until either
- /// `run_one_pass
- ()` returns `false`, or the event receiver hangs up. At this point, this
+    /// `run_one_pass()` returns `Some
+ (final_state)`, or the event receiver hangs up. At this point, this
    /// method calls the `event_stop_signaler.send
    //()` to terminate the receiver.
    ///
    /// This would run in a separate thread from the event receiver.
```

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-07] Misleading Implementation for `Signer#updateSigner()`

PR: 4788

Description

The `Signer#updateSigner()` implementation is currently left empty:

```
/// Refresh the next signer data from the given configuration data
fn update_signer(&mut self, _new_signer_config: &SignerConfig) {
    // do nothing
}
```

The Rust documentation for the function suggests that it should refresh the next signer data from the provided configuration data. However, the function does not perform this action.

Recommendation

Implement the `Signer#updateSigner()` function to ensure it updates the signer data as described.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-08] The StackerDB#send_message_bytes_with_retry() implementation could be simplified

PR: 4788

Description

The `StackerDB#send_message_bytes_with_retry()` implementation contains the following code:

```
let mut slot_version = if let Some(versions) = self.slot_versions.get_mut
    (msg_id) {
      if let Some(version) = versions.get(&slot_id) {
          *version
    } else {
          versions.insert(slot_id, 0);
          1 //@audit Why do we insert `0` but return `1`?
    }
  } else {
      let mut versions = HashMap::new();
      versions.insert(slot_id, 0);
      self.slot_versions.insert(*msg_id, versions);
      1
  };
```

It could be simplified for better readability and conciseness as follows:

```
let slot_version = self.slot_versions
        .entry(*msg_id)
        .or_insert_with(HashMap::new)
        .entry(slot_id)
        .or_insert(1);
```

Recommendation

Simplify the implementation as shown below:

```
) -> Result<StackerDBChunkAckData, ClientError> {
       let slot_id = self.signer_slot_id;
       loop {
-          let mut slot_version = if let Some
- (versions) = self.slot_versions.get_mut(msg_id) {
-                if let Some(version) = versions.get(&slot_id) {
-                     *version
-                } else {
-                     versions.insert(slot_id, 0);
-                     1
-                }
-          } else {
-                let mut versions = HashMap::new();
-                versions.insert(slot_id, 0);
-                self.slot_versions.insert(*msg_id, versions);
-                1
-          };
+          let slot_version = self.slot_versions
+                .entry(*msg_id)
+                .or_insert_with(HashMap::new)
+                .entry(slot_id)
+                .or_insert(1);
           let mut chunk = StackerDBChunkData::new
              (slot_id.0, slot_version, message_bytes.clone());
          chunk.sign(&self.stacks_private_key)?;
```

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-09] BACKOFF_MAX_INTERVAL should not exceed BACKOFF_MAX_ELAPSED

PR: 4788

Description

The `stacks-signer` client uses a backoff timer to delay retry attempts in `retry_with_exponential_backoff()`. However, the backoff timer is configured such that the maximum interval (`16384 ms`) exceeds the maximum elapsed time (`5 sec`). As a result, the delay can never reach the maximum interval since the timer will return `None` once the 5-second maximum elapsed time is reached.

Recommendation

Revisit the configuration for the backoff timer in the `stacks-signer` client.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-10] Improve `new_tenure` and `tenure_extended` Variable Naming

PR: 4827

Description

Within the `NakamotoChainState::append_block` function, the `new_tenure` variable represents a boolean indicating if the block corresponds to a new tenure, and the `tenure_extend` variable represents if the current tenure was extended.

Both variables can have better names to improve readability.

Recommendation

Rename `new_tenure` to `is_new_tenure` and `tenure_extend` to `is_tenure_extension` .

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-11] Sign Coordinator v0 Logging Discrepancies

PR: 4807

Description

The logging implemented in the `SignCoordinator::begin_sign_v0` function is, in some instances, misleading or could be improved:

1. If the block sighash does not match the response hash, a warning is shown indicating that an error will be returned. However, execution continues, and the signature is ignored. Either change the logging message or return an error, as done in the v1 implementation.
2. When failing to get the signer public key, an `NakamotoNodeError::SignerSignatureError` error is returned, but no error message is logged. Consider logging an error message. Similarly, log an error message when the signer entry is not found.
3. The current implementation does not log information about the current signing block on code paths. Log index information about the current block being signed, such as its block hash, at the beginning or end of the function.

Recommendation

Implement the recommended changes.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-12] Wrapping versus Saturating Reward Cycle Inconsistencies

PR: 4877

Description

Throughout the codebase, when working with a reward cycle, an increment is usually performed. However, this increment is handled differently with regards to wrapping or saturating the overflow values.

In some cases, the logic is to wrap the reward cycle, meaning if it reaches the maximum `uint64` value, it starts from 0. Examples of this can be found in `stacks-signer\\src\\v1\\signer.rs` and `stacks-signer\\src\\v1\\stackerdb_manager.rs` :

```
let next_reward_cycle = self.reward_cycle.wrapping_add(1);
// ... code ...
MessageSlotID::Transactions.stacker_db_contract
  (config.mainnet, config.reward_cycle.wrapping_add(1)),
MessageSlotID::Transactions.stacker_db_contract
  (is_mainnet, reward_cycle.wrapping_add(1)),
```

In other parts of the codebase, the logic is to saturate the reward cycle. An example can be found in `stacks-signer\\src\\runloop.rs` :

```
self.refresh_signer_config(current_reward_cycle.saturating_add(1));
```

Reaching the maximum `uint64` value for the reward cycle variable is an extreme corner case that requires careful consideration in theory. While it is unlikely to occur in practice, if it does, the behavior of the reward cycle variable (and other similar ones) should be consistent throughout the codebase.

Recommendation

Change all instances of `wrapping_add` to `saturating_add` to avoid reusing previous reward cycles in the extreme case that the value overflows.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-13] Move `force_send` Configuration to `ConnectionOptions`

PR: 4877

Description

Within the `Relayer` component, when relaying epoch3 blocks, the `relay_epoch3_blocks` function has a `force_send` configuration flag that relays epoch3 blocks even if already known.

The `relay_epoch3_blocks` function is currently only called with the flag set to `false` :

```
self.relay_epoch3_blocks
  (local_peer, sortdb, chainstate, accepted_blocks, false);
```

Having the `force_send` flag directly hardcoded makes code alterations more difficult, as developers are already expecting flags to be present in the connection options.

Recommendation

To improve system flexibility when configuring this option and to adhere to the single source of truth principle, move the `force_send` configuration to `ConnectionOptions` .

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-14] Improve NET Relayer Logging

PR: 4877

Description

Within the `stackslib\src\net\relay.rs::Relayer` component, there are several improvements that can be made to logging.

- In the `validate_nakamoto_blocks_push` function, the case where a sortition is not known is not logged, only silently skipped. Add an `info` or `debug` logging message.
- Consider changing `test_debug` logging to either `debug` or `info` logging level in the following cases:
  - ◊ When forwarding nakamoto blocks.
  - ◊ When discarding blocks because of staleness.
- Consider changing the `debug` level to `info` level when banning neighbor nodes, as it can be considered relevant information regardless of the debug state.
- Consider changing the logging level from `info` to `warn` when disregarding invalid Nakamoto blocks due to missing sortition.

Recommendation

Implement the recommended changes.

[QA-15] NakamotoBlocksData Consensus Deserialization Can Be Optimized

PR: 4877

Description

When a `NakamotoBlocksData` data structure is deserialized, a check is implemented to ensure that duplicate blocks are not allowed.

As the check is currently implemented, it first verifies if a block ID has already been processed, and if not, calls the `Hash::insert` function.

A simple and effective optimization can be achieved by directly verifying the return value of the `insert` call, as it returns `false` if the element was **not** inserted. If `false` is returned, halt execution.

By doing this, the `Hash::contains` operation can be completely removed.

The same type of optimization can be applied in `stackslib/src/net/mod.rs::consume_nakamoto_blocks`.

Recommendation

Implement the mentioned optimization. Example change:

```
// only valid if there are no dups
        let mut present = HashSet::new();
        for block in blocks.iter() {
-           if present.contains(&block.block_id()) {
+           if !present.insert(block.block_id()) {
                // no dups allowed
                return Err(codec_error::DeserializeError(
                   "Invalid NakamotoBlocksData: duplicate block".to_string(),
                ));
            }
-
-           present.insert(block.block_id());
}
Ok(NakamotoBlocksData { blocks })
```

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-16] Use Constants Instead of Magic Numbers in `RPCRequestHandler`

PR: 4902

Description

Using constants instead of hardcoding values generally improves code readability. In the `RPCRequestHandler::try_handle_request` function, if there is an error when collecting the response from the node, a `400 Bad Request` error is returned. However, the error code `400` is hardcoded.

Recommendation

Either import and use `http::StatusCode::BAD_REQUEST` or define a local constant for better code clarity.

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-17] Improve Logging in Nakamoto Chainstate Module Code

PR: 4902

Description

Within the context of PR 4930, there are several locations where the debug logging can be improved:

1. The log message for rejecting the storage of a duplicate block due to signer weight difference mentions the case that `since it has less signing power` . However, this execution flow is reached even when the signing power is equal. Change the message to `since it has less or equal signing power` .
2. When retrieving a block header by coinbase height via the `get_header_by_coinbase_height` function, there is no warning message when exiting the function without having found the header height. Add a warning message in this case.

Recommendation

Implement the mentioned changes.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-18] NakamotoStagingBlocksConnRef::has_children is never used

PR: 4930

Description

In the `staging_blocks.rs` source file, the `NakamotoStagingBlocksConnRef::has_children` function is never used, either internally or externally.

Recommendation

Consider reusing it in an appropriate context or removing it.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-19] Indistinguishable NakamotoBlockBuilder::load_tenure_info Error Messages

PR: 4879

Description

Within the `NakamotoBlockBuilder::load_tenure_info` function, there are two distinct operations that share the same error message in case of issues. Both when determining the coinbase height from the POX calculation (`coinbase_height_of_calc`) and when retrieving the active reward set information (`active_reward_set`), the same error message is used:

```
).map_err(|e| {
      warn!(
              "Cannot process Nakamoto block: could not load reward set that elected the blo
          "err" => ?e,
      );
      Error::NoSuchBlockError
})?;
```

This will cause confusion if one of the two cases ever appears.

Recommendation

When retrieving the `coinbase_height_of_calc` value, change the error message to a more appropriate one, such as: `"Cannot process Nakamoto block: could not retrieve coinbase POX coinbase POX height of the elected block"` .

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-20] mod::NakamotoChainState::check_pox_bitvector can be simplified

PR: 4879

Description

The `NakamotoChainState::check_pox_bitvector` function initially checks if a tenure block commit has an empty treatment vector. If it does not, the function continues with its logic; otherwise, it exits.

Currently, this is implemented as:

```
if !tenure_block_commit.treatment.is_empty() {
  // ... a lot of code ...
}
Ok(())
```

Since there is a lot of code within the `if-true` branch, the execution can be short-circuited by negating the logic. This reduces code clutter and simplifies the `if` statement:

```
if tenure_block_commit.treatment.is_empty() {
  Ok(())
}
// ... a lot of code ...
Ok(())
```

Recommendation

Apply the indicated recommendation.

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-21] Use Descriptive Variable Names

PR: 4827

Description

Avoid using single-letter variable names to enhance code readability and quality. This recommendation applies to several PRs:

4902

- In the `Signer::process_event` function, the `b` variable.

4877

- The `w` variable in `RelayerStats::sample_neighbors`.

4877

- Variables named `h` in `RelayPayload` and `RelayerStats`.
- `ch` and `bh` suffixes in relayer.rs#L682.
- `pkx` suffix in `relayer.rs#L700`.

Recommendation

Use more descriptive names for the mentioned variables.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-22] Unused Imports

PR: 4788

Description

The codebase currently contains numerous unused imports across almost all source files. The mentioned occurrences are for reference and do not cover all instances in the codebase.

Recommendation

Consider removing the unused imports from your codebase to improve quality and readability.

Additionally, it is recommended to add a lint job to the CI pipelines to ensure that no unused imports are introduced with each new PR.

The code line references are from the last commit of PR4877. However, this issue is present across all branches/PRs of the codebase.

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-23] Inconsistent Ordering of Match Cases with StacksMessageType Throughout the Code

PR: 4877

Description

Within the codebase, there are slight differences in the order in which some match cases check for various `StacksMessageType`s.

For example, in p2p.rs#L1386-L1427, the ordering is as follows:

Block → Microblock → Nakamoto block → Transaction

From a logical standpoint, this is correct (going from the largest entity downwards).

However, in src/net/mod.rs#L1590-L1636, the ordering is different:

Block → Microblock → Transaction → Nakamoto block

In this case, the ordering of the match cases is not logically sorted, which slightly increases the difficulty in reasoning about this code snippet.

Recommendation

Reorder the cases of the match block in question in the following way:

CONTENTS


```
tacksMessageType::Microblocks(mblock_data) => {
            if let Some(mblocks_msgs) = self.pushed_microblocks.get_
mut(&neighbor_key) {
                mblocks_msgs.push((message.relayers, mblock_data));
            } else {
                self.pushed_microblocks.insert(
                    neighbor_key.clone(),
                    vec![(message.relayers, mblock_data)],
                );
            }
        }
+       StacksMessageType::NakamotoBlocks(block_data) => {
+           if let Some(nakamoto_blocks_msgs) =
+               self.pushed_nakamoto_blocks.get_mut(&neighbor_key)
+           {
+               nakamoto_blocks_msgs.push((message.relayers, block_data));
+           } else {
+               self.pushed_nakamoto_blocks
+                   .insert(neighbor_key.clone(), vec![(message.relayers,
block_data)]);
+           }
+       }
        StacksMessageType::Transaction(tx_data) => {
            if let Some(tx_msgs) = self.pushed_transactions.get_mut(&neighbor_
key) {
                tx_msgs.push((message.relayers, tx_data));
            } else {
                self.pushed_transactions
                    .insert(neighbor_key.clone(), vec![(message.relayers, tx_
data)]);
            }
        }
-       StacksMessageType::NakamotoBlocks(block_data) => {
-           if let Some(nakamoto_blocks_msgs) =
-               self.pushed_nakamoto_blocks.get_mut(&neighbor_key)
-           {
-               nakamoto_blocks_msgs.push((message.relayers, block_data));
-           } else {
-               self.pushed_nakamoto_blocks
-                   .insert(neighbor_key.clone(), vec![(message.relayers,
block_data)]);
-           }
-       }
```

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-24] Incorrect Sortition DB Schema 4 SQL Commands

PR: 4879

Description

The `SORTITION_DB_SCHEMA_4` from `stackslib\\src\\chainstate\\burn\\db\\sortdb.rs` contains a typo when creating the `ast_rule_heights` table:

```
const SORTITION_DB_SCHEMA_4: &'static [&'static str] = &[
    r#"
    CREATE TABLE delegate_stx (
        txid TEXT NOT NULL,
        vtxindex INTEGER NOT NULL,
        block_height INTEGER NOT NULL,
        burn_header_hash TEXT NOT NULL,
        sender_addr TEXT NOT NULL,
        delegate_to TEXT NOT NULL,
        reward_addr TEXT NOT NULL,
        delegated_ustx TEXT NOT NULL,
        until_burn_height INTEGER,
        PRIMARY KEY(txid, burn_header_Hash)
    );"#,
    r#"
    CREATE TABLE ast_rule_heights (
        ast_rule_id INTEGER PRIMARY KEY NOT NULL,
        block_height INTEGER NOT NULL
    );"#,
];
```

The `ast_rule_id INTEGER PRIMAR KEY NOT NULL,` command is invalid because of a typo in `PRIMAR` , which should be `PRIMARY` .

Recommendation

Resolve the typo. It is also recommended to always use the same letter case when referencing identifiers since some databases are case-sensitive. The `PRIMARY KEY(txid, burn_header_Hash)` has an uppercase `Hash` while the column ID is lowercase `burn_header_Hash)` .

CONTENTS

ClarityAlliance
Security Review

Nakamoto Upgrade

[QA-25] Improve Nakamoto Node Miner Thread Debug Logging

PR: 4827

Description

There are a few improvements that can be made to the `miners::BlockMinerThread` debug logging implementation as of the latest code changes:

1. In the `run_miner` function, also debug print the `self.burn_election_block.consensus_hash`, which was added in `PR #4827`.
2. In the `make_tenure_start_info` function, split the `debug!` macro across multiple lines for better code readability.

Recommendation

Implement the mentioned changes.

[QA-26] Continue Tenure Directive Logging Ambiguities

PR: 4827

Description

A tenure continuation directive that fails because the previous tenure could not be stopped is incorrectly considered a successful tenure continuation. This issue arises because the `relayer::continue_tenure` function, which is called from the `relayer::handle_sortition` function, incorrectly returns an `Ok` response instead of an error when it fails to stop the previous tenure.

```
if let Err(e) = self.stop_tenure() {
    error!("Relayer: Failed to stop tenure: {:?}", e);
    return Ok(());
}
```

Subsequently, in the `relayer::handle_sortition` function, the debug message `("Relayer: successfully handled continue tenure.");` is logged instead of an error message.

The same issue appears in the `relayer::continue_tenure` function if starting a new tenure fails. An error message is displayed, followed by an `Ok` response.

```
// ... code ...
Err(e) => {
error!("Relayer: Failed to start new tenure: {:?}", e);
}
Ok(())
```

In each case, there will be error messages from within the `continue_tenure` function, followed by a success message from the `handle_tenure` function. These ambiguous logging messages increase debugging time.

There is also a slight redundancy in the code, as the error arm in the `relayer::continue_tenure` response returns a `false` boolean. This is redundant because the return value of the parent `match` operation is not caught, since `handle_sortition` always returns true.

Recommendation

Modify the error messages to better describe the execution flow. Returning an error in both mentioned instances would also clarify the code logic. Additionally, remove the return from the error branch in `continue_tenure`.