ClarityAlliance

Stacking DAO PoX-5 Security Review

Conducted by:
Kristian Apostolov, Silverologist

August 13th, 2026

Contents

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.

2. Disclaimer

This report is not, and should not be considered, an endorsement or disapproval of any project, team, product, or asset, nor does it reflect on their economics, value, business model, or legal compliance. It does not provide any warranty regarding the absolute bug-free nature or functionality of the analyzed technology.

Nothing in this report should be used to make investment or participation decisions. It does not constitute investment advice. Instead, it reflects an extensive assessment process intended to help clients improve code quality and reduce the inherent risks associated with cryptographic tokens and blockchain systems.

Blockchain technology presents ongoing and significant risk, and each company or individual remains responsible for their own due diligence and security posture. Clarity Alliance aims to reduce attack vectors and technological uncertainty but does not guarantee the security or performance of any system we review.

All assessment services are subject to dependencies and active development. Your access to and use of any services, reports, or materials is at your sole risk on an as-is and as-available basis.

Cryptographic tokens are emergent technologies with high technical uncertainty, and assessment results may include false positives, false negatives, or other unpredictable outcomes. Smart contracts may depend on multiple external parties, remain vulnerable to internal or external exploitation, and may carry elevated risks if owner privileges remain active. Accordingly, Clarity Alliance does not guarantee the explicit security of any audited smart contract, regardless of the reported verdict.

3. Introduction

A time-boxed security review of the Stacking DAO Protocol Rewrite, where Clarity Alliance reviewed the protocol's redesigned smart contract architecture and provided recommendations to improve its security, correctness, and resilience. The review covered all smart contracts included in the rewritten protocol, representing a comprehensive redesign of the StackingDAO protocol.

4. About Stacking DAO

What is Stacking DAO, and how does it work?

Stacking DAO is the STX Stacking infrastructure powerhouse for the most prominent Bitcoin L2. The protocol currently offers 3 STX Stacking services:

- **stSTX**: A liquid representation of stacked STX that accrues in value in STX as Stacking rewards are auto-compounded daily. It can also be used across DeFi to earn additional yield and points.
- **stSTXbtc**: A liquid stacking token backed 1-to-1 with STX, and holders receive sBTC rewards daily that can be claimed at any moment. stSTXbtc can also be used across Stacks dApps.
- **Native Stacking**: Delegate STX and earn BTC rewards with zero fees while STX are locked during the two-week Stacking cycles.

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

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

6. Security Assessment Summary

Scope

Repository: `StackingDAO/smart-contracts`

The scope of the review covered all the smart contracts in the repository.

Initial Commit Reviewed

`34911b5348da2e2375cffadb933d898d6c81e6d7`

Final Commit Reviewed

`4bdb84b65b09274d7325ff1c6dae71c47adba4b7`

The audited contracts were reviewed in the *StackingDAO/smart-contracts* repository. In the final reviewed commit, the repository had been renamed to *StackingDAO/stackingdao-smart-contracts*.

7. Executive Summary

Over the course of the security review, Kristian Apostolov, Silverologist engaged with Stacking DAO to review Stacking DAO. In this period of time a total of **31** issues were uncovered.

Protocol Summary

Protocol Name	Stacking DAO
Repository	StackingDAO/smart-contracts
Report Date	August 13th, 2026

Findings Count

Severity	Amount
Critical	7
High	5
Medium	10
Low	8
QA	1
Total Findings	**31**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Native Pool Credits Rewards for Stake Delegated to an Arbitrary Signer Manager	Critical	Resolved
[C-02]	Native Pool Distributes Historical Rewards Using Current Weights	Critical	Resolved
[C-03]	Pending Scheduled Withdrawals Allow New Deposits to Mint Excess Shares	Critical	Resolved
[C-04]	Stale Price Ratio Allows Deposits to Capture Pending Rewards	Critical	Resolved
[C-05]	PoX-5 Stake Updates Allow Users to Retain Native-Pool Reward Weight After Leaving the Native Signer	Critical	Resolved
[C-06]	Pending stSTXbtc Withdrawals Depress the stSTX Ratio and Allow Share Inflation	Critical	Resolved
[C-07]	Native Pool Reimplements Default Signer Reward Distribution, Causing Reward Loss	Critical	Resolved
[H-01]	Core Is Vulnerable to a First-Depositor Attack	High	Resolved
[H-02]	Permissionless Dust Funding Allows Indefinite Extension of STX Reward Streams	High	Resolved
[H-03]	Stale stSTXbtc Tracker Allows Late Depositors to Capture Accrued sBTC Rewards	High	Resolved

[H-04]	stSTXbtc Principal Is Unallocated	High	Resolved
[H-05]	Incorrect and Missing Usage of with-stacking	High	Resolved
[M-01]	Late Reward Tracker Exclusion Leaves Phantom Supply and Locks Rewards	Medium	Resolved
[M-02]	Hard 100x Ratio Cap Can Block Reward Split Updates	Medium	Resolved
[M-03]	New Withdrawal Fee Is Applied Retroactively to Initiated Withdrawals	Medium	Resolved
[M-04]	SIP-010 Compliance Issues	Medium	Resolved
[M-05]	SIP-009 Compliance Issues	Medium	Resolved
[M-06]	DAO Setters Authorizing tx-sender Enable Phished Admin Calls	Medium	Resolved
[M-07]	Incorrect Rounding Direction in stBTC and stSTX Deposits	Medium	Resolved
[M-08]	Native Signer Manager Access Control Can Be Bypassed	Medium	Resolved
[M-09]	stSTX Scheduled Withdrawals Can Drain STX Earmarked for stSTXbtc	Medium	Resolved
[M-10]	stSTXbtc Withdrawal Fees Accrue to stSTX Holders	Medium	Resolved
[L-01]	Full Boost Collapses Positive TVL to a Discontinuous 100% stBTC Reward Split	Low	Resolved
[L-02]	DAO Can Brick Itself	Low	Resolved

[L-03]	Reward Tracker Exclusion Creates Phantom Supply and Strands Rewards	Low	Resolved
[L-04]	Native Pool Cannot Enforce DAO-Gated Unstake	Low	Resolved
[L-05]	Lack of Timelock on New DAO Admins Allows Instant Drain on Principal Compromise	Low	Acknowledged
[L-06]	sDAO Staking Global Pause Blocks Exits and Reward Claims	Low	Resolved
[L-07]	Scheduled Withdrawal Cooldown May Mature Before Reserve Liquidity Is Available	Low	Partially Resolved
[L-08]	Withdrawal Fees Cause Immediate Price Ratio Increases	Low	Acknowledged
[QA-01]	Deprecated Contracts Can Be Removed	QA	Resolved

8. Findings

8.1. Critical Findings

[C-01] Native Pool Credits Rewards for Stake Delegated to an Arbitrary Signer Manager

Location:

- native-pool-v1.clar#L31

Description `native-pool-v1::delegate` allows the caller to supply an arbitrary PoX-5 signer manager:

```
(define-public (delegate
    (signer-manager <signer-manager-trait>)
    (amount-ustx uint)
    (num-cycles uint)
  )
```

The function then stakes the user's STX via the provided signer manager and credits the user with native-pool reward weight in `reward-tracker-template-v1`.

This breaks the native pool accounting invariant. The native pool tracker should only include users whose STX is staked behind `native-pool-signer-manager`, because only that signer manager forwards claimed PoX-5 rewards into `reward-tracker-template-v1`.

An attacker can instead pass a different valid signer manager. Their STX is genuinely staked, but not behind the native-pool signer manager. Despite this, they are still credited with native-pool weight. When the real native-pool signer manager claims rewards, those rewards are shared with the attacker. Separately, the attacker may also receive rewards from their own signer manager.

As a result, the attacker can siphon rewards from honest native-pool participants without contributing their signer manager's rewards to the native-pool tracker.

Recommendation

`native-pool-v1` should not accept an arbitrary signer manager for delegation. Instead, it should use (or hardcode) the expected `native-pool-signer-manager` and prevent users from providing an alternative signer manager.

[C-02] Native Pool Distributes Historical Rewards Using Current Weights

Location:

- native-pool-signer-manager.clar#L77

Description `native-pool-signer-manager::claim-rewards` claims rewards for a specific historical PoX reward cycle, but then forwards those rewards to `native-pool-tracking-v1::add-rewards`, which distributes them based on the tracker's *current* balances.

As a result, native pool rewards are not distributed according to who actually staked during the rewarded cycle. A user can delegate after a cycle's rewards have already been earned and calculated, but before those rewards are claimed into the tracker. Once they delegate, their current tracker weight is included, allowing them to receive a share of rewards from a prior cycle.

For example:

- An honest user delegates before cycle 1 and is the only staker for cycle 1.
- Cycle 1 rewards are funded and calculated.
- A late joiner delegates for cycle 2.
- The native signer claims cycle 1 rewards into the tracker.
- The tracker splits the cycle 1 rewards between the honest user and the late joiner.

This allows late delegators to capture rewards that should have been distributed only to users who participated in the rewarded cycle.

Recommendation

Update native-pool reward accounting to distribute rewards on a per-reward-cycle basis, rather than using the tracker's current total balances.

[C-03] Pending Scheduled Withdrawals Allow New Deposits to Mint Excess Shares

Location:

- stacking-dao-core-stbtc-v1.clar#L64

Description When a user initiates a scheduled stBTC withdrawal, `stacking-dao-core-stbtc-v1::init-withdraw` locks in the user's sBTC claim, reserves the corresponding sBTC, and escrows the user's stBTC in the core contract. However, the escrowed stBTC is not burned until the withdrawal is finalized.

As a result, during the cooldown period, the stBTC ratio continues to account for:

- the sBTC already reserved for the pending withdrawal, and
- the escrowed stBTC supply, which should no longer participate in future rewards.

If rewards enter the reserve while the withdrawal is pending, the reported ratio becomes lower than the true ratio for active holders. A new deposit made during this window will therefore mint too many stBTC shares. Once the pending withdrawal is finalized and the escrowed stBTC is burned, the ratio increases, effectively transferring value to the new depositor that should have accrued to active holders.

For example:

- Alice and Bob each deposit 1 sBTC and receive 1 stBTC.
- Bob initiates a scheduled withdrawal, fixing a 1 sBTC claim while his 1 stBTC remains in the total supply.
- A 1 sBTC reward enters the reserve.
- Before Bob's withdrawal is finalized, Charlie deposits 1.5 sBTC and receives 1 stBTC.
- After Bob's withdrawal is finalized, Charlie withdraws.

Because the sBTC:stBTC rate increases when Bob's withdrawal is finalized, Charlie can realize an almost immediate profit, capturing rewards that should have belonged to Alice.

Recommendation

Exclude pending scheduled withdrawals from active ratio accounting.

For example, subtract pending-withdrawal assets and escrowed withdrawal shares from the ratio calculations.

[C-04] Stale Price Ratio Allows Deposits to Capture Pending Rewards

Location:

- stacking-dao-core-stx-v1.clar#L45
- stacking-dao-core-stx-v1.clar#L63
- stacking-dao-core-stx-v1.clar#L124

Description

`stacking-dao-core-stx-v1::deposit` mints stSTX using `data-stx-v1::get-stx-per-ststx`.

However, the ratio returned by the getter only reflects STX already held by, or accounted for in, `stx-reserve`. It does not include STX rewards that have accrued in `rewards-stx-v1` but have not yet been crystallized via `process-rewards`.

Because `deposit` does not call `rewards-stx-v1::process-rewards` before calculating the mint ratio, a depositor can enter at a stale (lower) ratio after rewards have accrued for existing holders. The depositor can then call `process-rewards` directly, causing the accrued STX to be moved into the reserve and increasing the ratio, while the depositor's newly minted shares also benefit from that increase.

As a result, a new depositor can capture a portion of rewards that should belong to existing stakers.

For example:

1. Alice deposits 10 STX and receives 10 stSTX.
2. A 10 STX reward is streamed to stSTX holders.
3. The full 10 STX reward accrues, but no one calls `process-rewards`, so `stx-reserve` still reports only 10 STX backing.
4. An attacker deposits 10 STX at the stale 1:1 ratio and receives 10 stSTX. If rewards had been crystallized first, the ratio would be 2:1 and the attacker would receive only 5 stSTX.

5. The attacker calls `rewards-stx-v1::process-rewards`, moving the vested 10 STX into the reserve.
6. The reserve now has 30 STX backing 20 stSTX, so the ratio is 1.5.
7. The attacker withdraws 10 stSTX via idle withdrawal and receives 14.85 STX after the 1% fee.
8. The attacker profits 4.85 STX, taken from Alice's already vested reward.

A similar issue exists for idle withdrawals and delayed withdrawal initiations: users may not receive accrued (but not yet crystallized) rewards due to the stale ratio.

The stBTC product is affected in the same way, as its price ratio does not include pending rewards from `rewards-v7::pending-rewards`.

Recommendation

At the start of the `deposit`, `init-withdraw`, and `withdraw-idle` functions for the stSTX product, call `rewards-stx-v1::process-rewards` before reading the stSTX price ratio. For the stBTC product, call `rewards-v7::process-rewards` instead.

[C-05] PoX-5 Stake Updates Allow Users to Retain Native-Pool Reward Weight After Leaving the Native Signer

Location:

- native-pool-v1.clar#L30

Description `native-pool-v1::delegate` registers a user, refreshes their reward weight in `native-pool-tracking-v1`, and then forwards the delegation to `pox-5::stake`.

After delegating, however, the user can interact with `pox-5` directly.

Specifically, a user can delegate through `native-pool-v1` to the native signer manager and then call `pox-5::stake-update` directly to move their PoX-5 stake to a different signer manager. This is possible because the native signer manager's `checkpoint-staker` hook is a no-op. As a result, `native-pool-v1` is not invoked and `native-pool-tracking-v1` is not refreshed.

Consequently, the user no longer has PoX-5 stake delegated behind the native-pool signer manager, but they retain their previously recorded native-pool reward weight.

When `native-pool-signer-manager::claim-rewards` later claims native signer rewards into `native-pool-tracking-v1`, those rewards are still distributed to the user who has already left. This enables an attacker to siphon sBTC rewards from honest native-pool delegators while also earning rewards from their new signer.

Recommendation For the `pox-5` version currently used in the codebase, this scenario can be mitigated via the `checkpoint-staker` hook in the native signer manager. However, this hook has been removed in newer versions of `pox-5`. As a result, signer managers are no longer notified when a user moves delegated STX elsewhere and cannot automatically update native-pool tracking.

Given this limitation, `native-pool-tracking-v1` reward weights cannot remain accurate if maintained independently. Instead, the protocol should rely on the reward-weight accounting already maintained by the `pox-5` contract.

Accordingly, the native signer manager should call `pox-5::claim-rewards` each cycle to pull reward sBTC to itself, then call `pox-5::claim-staker-rewards-for-signer` and transfer each staker's earned rewards to them.

[C-06] Pending stSTXbtc Withdrawals Depress the stSTX Ratio and Allow Share Inflation

Location:

- stacking-dao-core-ststxbtc-v1.clar#L63

Description stSTXbtc deposits increment `stx-reserve::stx-for-ststxbtc`, earmarking STX 1:1 for stSTXbtc principal. When a user initiates a scheduled stSTXbtc withdrawal, `stacking-dao-core-ststxbtc-v1::init-withdraw` also increments `stx-for-withdrawals`, but does not decrement `stx-for-ststxbtc` until the withdrawal is finalized.

During the cooldown period, the same STX is effectively counted in both counters. Because `data-stx-v1::get-stx-per-ststx` subtracts `stx-for-withdrawals + stx-for-ststxbtc` from the total STX backing, the backing for stSTX is understated and the stSTX ratio is artificially depressed.

This enables an attacker to mint excess stSTX.

For example:

- Alice deposits 100 STX into stSTX.
- An attacker deposits 10 STX into stSTXbtc and initiates a scheduled withdrawal, depressing the stSTX ratio from 1.0 to 0.9.
- The attacker deposits 90 STX into stSTX and receives 100 stSTX instead of the fair 90 stSTX.
- After finalizing the stSTXbtc withdrawal, the attacker redeems the inflated 100 stSTX for 95 STX.

In total, the attacker deposits 100 STX and withdraws 105 STX. Existing stSTX holders are left with only 95 STX backing a 100 stSTX supply.

Recommendation

When initiating a stSTXbtc withdrawal, move the amount from `stx-for-ststxbtc` to `stx-for-withdrawals` rather than creating an overlapping

reservation. `init-withdraw` should atomically release the stSTXbtc earmark when it creates the withdrawal reservation.

[C-07] Native Pool Reimplements Default Signer Reward Distribution, Causing Reward Loss

Location:

- native-pool-signer-manager.clar

Description The current `native-pool-signer-manager` claims aggregate rewards from `pox-5` and transfers them to `native-pool-v1`. `native-pool-v1` then distributes those rewards to individual stakers by reading PoX-5 per-cycle staking weights and calculating each user's proportional share.

This approach duplicates reward distribution functionality already provided by the default PoX-5 signer manager implementation. In the default design, the signer manager claims aggregate rewards from `pox-5`, and stakers claim their individual rewards directly via `pox-5::claim-staker-rewards-for-signer`, relying on PoX-5's native reward-settlement accounting.

As a result, the native-pool design introduces an additional distribution contract, local per-cycle distribution state, local double-claim tracking, and new failure modes—without a clear accounting advantage.

For example, `native-pool-v1` stores only one reward distribution per PoX reward cycle and rejects any second distribution for the same cycle with `ERR_ALREADY_DISTRIBUTED`. This is incompatible with PoX-5 reward accounting because `pox-5::calculate-rewards` runs on distribution cycles, and there are two distribution cycles per PoX reward cycle. Consequently, a single PoX reward cycle may receive multiple incremental reward calculations.

If the native pool claims rewards after the first distribution calculation, the cycle is marked as distributed in `native-pool-v1`. When the second distribution calculation later generates additional rewards for the same PoX reward cycle, `native-pool-signer-manager::claim-rewards` cannot forward them to `native-pool-v1` because the subsequent `distribute-cycle-reward` call reverts. Those later rewards remain locked.

Recommendation Remove the reward distribution logic from `native-pool-v1`. If a protocol-owned wrapper is still desired for UX, limit it to convenience calls around PoX-5 staking actions.

Replace `native-pool-signer-manager` with a minimized version of the default signer manager provided by StacksLabs, with the following changes:

- Add the DAO-enabled check to `validate-stake!`.
- Replace the local `update-admin` model with DAO-controlled authorization.
- Remove the L1 withdrawal logic and fee logic, as these components are not used by StackingDAO.

8.2. High Findings

[H-01] Core Is Vulnerable to a First-Depositor Attack

Location:

- stacking-dao-core-stbtc-v1.clar#L46
- stacking-dao-core-stx-v1.clar#L45

Description
The stBTC deposit flow mints shares based on the current reserve ratio:

```
stbtc-amount = sbtc-amount * DENOMINATOR_8 / ratio
```

There is no `min-shares-out` parameter and no protection against a first depositor manipulating the ratio when the share supply is very low.

As a result, an attacker can execute a classic first-depositor attack, with a minor adjustment to account for the requirement that at least 1 sat of shares must be minted for the transaction to succeed.

An attacker can become the first stBTC holder by making a tiny initial deposit and then waiting for a victim deposit. Immediately before the victim's transaction, the attacker front-runs by donating sBTC directly to the reserve, inflating the sBTC-per-stBTC ratio. This inflation can be calibrated so that the victim would otherwise receive nearly 2 sats of shares, but due to rounding down receives only 1.

The rounded-down amount is effectively redistributed across existing holders, benefiting the attacker (and partially the victim).

Recommendation
Seed the initial deposit with permanent dead shares. On the first deposit, mint a fixed portion of the corresponding shares to a burn address so they are permanently unrecoverable, and mint only the remaining shares to the first depositor.

The seed should be large enough to prevent share-ratio manipulation from making first-depositor attacks profitable. A common approach is to use 1,000 units of dead shares.

Additionally, consider adding a `min-shares-out` slippage parameter to deposits and reverting if the number of minted shares is below the user's specified minimum.

[H-02] Permissionless Dust Funding Allows Indefinite Extension of STX Reward Streams

Location:

- rewards-stx-v1.clar#L58

Description `rewards-stx-v1::add-rewards` is permissionless and immediately calls `process-rewards` after transferring STX from the caller:

```
(try! (stx-transfer? amount tx-sender current-contract))
  (process-rewards)
```

`process-rewards` is also public and permissionless.

When `process-rewards` detects newly received STX, it folds the currently queued stream together with the new amount and restarts distribution over a fresh `RELEASE_WINDOW_BLOCKS` window:

```
new-total = queued + new-to-fold
new-end = burn-block-height + RELEASE_WINDOW_BLOCKS
new-per-block = new-total / RELEASE_WINDOW_BLOCKS
```

As a result, any user can extend an active reward stream at negligible cost.

For example, assume 2,100 STX is scheduled to stream over 2,100 blocks. After 1,000 blocks, an attacker calls `add-rewards(1)`. The contract first releases the accrued 1,000 STX, then combines the remaining 1,100 STX with the attacker's 1 STX and redistributes the total over a new 2,100-block window.

By repeating this behavior, an attacker can continuously delay rewards from reaching `stx-reserve`, suppressing the stSTX ratio increase and distorting deposit and withdrawal values.

`rewards-v7` exhibits the same vulnerability pattern. An attacker can create `new-to-fold > 0` by transferring a minimal amount of sBTC directly to

`rewards-v7`, then calling `process-rewards` to re-fold the remaining rewards into a new full reward window.

Recommendation

Implement a keeper authorization model similar to `signer-payout-v1`, where only the keeper is permitted to call `add-rewards`.

Step 1 (crystallization) should remain publicly accessible, while Step 2 (folding) should be permissioned and restricted to the keeper.

[H-03] Stale stSTXbtc Tracker Allows Late Depositors to Capture Accrued sBTC Rewards

Location:

- ststxbtc-token.clar

Description stSTXbtc rewards are distributed via `ststxbtc-tracking-v1`. When `rewards-v7::process-rewards` crystallizes the stSTXbtc reward share, it calls `ststxbtc-tracking-v1::add-rewards`, which allocates the reward amount across the tracker's current total supply.

However, stSTXbtc balances can change before previously accrued rewards are crystallized. A depositor can wait until a `rewards-v7` stream has fully accrued, deposit STX to mint stSTXbtc at a 1:1 rate, and then call `rewards-v7::process-rewards`. Because the tracker uses the post-deposit supply when distributing the already-accrued reward, the new depositor receives a portion of rewards that accrued before they held any stSTXbtc.

For example:

- Alice holds 10 stSTXbtc throughout a fully accrued 1 sBTC reward stream.
- Before rewards are processed, the attacker deposits 10 STX and receives 10 stSTXbtc.
- The attacker processes rewards. The tracker distributes 1 sBTC over 20 stSTXbtc, resulting in 0.5 sBTC for Alice and 0.5 sBTC for the attacker.

If rewards had been processed before the attacker's deposit, Alice would have received the full 1 sBTC.

This issue also applies to transfers and exits, since those operations refresh tracker balances as well.

Recommendation

Invoke `rewards-v7::process-rewards` at the start of `ststxbtc-token::mint-for-protocol`, `transfer`, and `burn-for-protocol`, before any balance changes and before calling `refresh-wallet`.

[H-04] stSTXbtc Principal Is Unallocated

Location:

- stacking-dao-core-ststxbtc-v1.clar

Description stSTXbtc is an STX-in product intended to earn cumulative sBTC rewards through the PoX-5 STX-only staking track. It is explicitly listed alongside stSTX and the native pool as using STX-only staking rather than protocol bonds.

However, the current implementation effectively treats deposited stSTXbtc principal as permanently reserved liquidity.

On deposit, `stacking-dao-core-ststxbtc-v1` transfers STX into `stx-reserve`, mints stSTXbtc 1:1, and calls:

```
(stx-reserve::lock-stx-for-ststxbtc stx-amount)
```

`stx-reserve::get-stx-available` then subtracts `stx-for-ststxbtc`, and both bond collateral and STX-only staking draw from this same available balance via `request-stx-to-stack`. As a result, stSTXbtc backing is excluded not only from bonds, but also from STX-only PoX-5 staking.

Consequently, stSTXbtc holders may receive a configured share of sBTC rewards through `rewards-v7` and `ststxbtc-tracking-v1`, even though their deposited STX is not actually deployed to generate those rewards. Economically, this causes the product to be subsidized by other deployed capital, and the reward split calculator may allocate rewards to a TVL leg that is not contributing PoX-5 stake.

Recommendation Allow stSTXbtc idle principal to be deployed in STX-only staking.

[H-05] Incorrect and Missing Usage of with-stacking

Location:

- stbtc-staker-template-v1.clar#L40
- stbtc-staker-template-v1.clar#L85
- stx-staker-template-v1.clar#L71

Description
The protocol's staker wrappers apply incomplete `as-contract?` allowances around PoX-5 calls that lock STX or update an existing STX lock.

In `stbtc-staker-template-v1`, both `bond-sbtc` and `rollover-bond` call `pox-5::register-for-bond` while only providing an sBTC `with-ft` allowance. However, `register-for-bond` also locks the staker contract's STX collateral. PoX STX locks are not standard STX transfers covered by `with-stx`; instead, Stacks Core records them as stacking asset actions, which must be authorized via `with-stacking`.

Similarly, `stx-staker-template-v1::stake-update-stx` wraps `pox-5::stake-update` with `with-stacking amount-increase`. PoX-5 (and Stacks Core) records the *new total locked amount*, not just the incremental increase. As a result, an update that increases an existing stake may exceed the provided allowance even when `amount-increase` itself is correct.

Consequently, these PoX-5 calls may fail due to stacking asset allowance enforcement, even when the protocol has sufficient liquidity.

Recommendation

- For `register-for-bond`, include a `with-stacking` allowance covering the full STX collateral amount being locked.
- For `stake-update-stx`, authorize the new total locked amount rather than `amount-increase`.

8.3. Medium Findings

[M-01] Late Reward Tracker Exclusion Leaves Phantom Supply and Locks Rewards

Location:

- sdao-staking-sbtc-v1.clar#L158

Description `sdao-staking-sbtc-v1` streams sBTC rewards over time using a cumulative reward-per-stake accumulator. If all sDAO stakers exit while a reward stream is still active, the remaining streamed rewards can become unassigned.

When `total-staked == 0`, `calculate-cumm-reward-per-stake` returns the current accumulator unchanged. However, a later call to `increase-cumm-reward-per-stake` still advances `last-reward-increase-block`. As a result, elapsed stream time is skipped without attributing the corresponding sBTC to any staker.

For example, Alice stakes during a 1 sBTC stream, receives only the vested 40%, and then fully exits. The stream later completes while `total-staked` remains zero. When Bob subsequently stakes and triggers the accumulator update, the previously elapsed time is effectively ignored, leaving the remaining 0.6 sBTC locked in `sdao-staking-sbtc-v1`.

Because `sdao-staking-sbtc-v1` has no recovery mechanism, these unassigned rewards have no path to be reclaimed.

Recommendation

Introduce an explicit `unassigned-rewards` accounting bucket in `sdao-staking-sbtc-v1`. In `increase-cumm-reward-per-stake`, compute `block-diff` in the same way as `calculate-cumm-reward-per-stake`. Then, if `total-staked` is 0, accrue the skipped rewards to `unassigned-rewards`:

```
unassigned-rewards += rewards-per-block * block-diff
```

Then, add a DAO-gated `sweep-unaccounted-rewards` function that calls `increase-cumm-reward-per-stake` and withdraws `unassigned-rewards` to a DAO-controlled principal. Alternatively, update `add-rewards` to automatically roll any pending unassigned rewards into the next reward cycle whenever it is executed.

[M-02] Hard 100x Ratio Cap Can Block Reward Split Updates

Location:

- reward-split-calculator-v1.clar#L128

Description `reward-split-calculator-v1::compute-and-apply` rejects any `r-current` value greater than `r-target * 100`:

```
(asserts! (<= r-current (* (var-get r-target) u100)) ERR_INVALID_PARAMS)
```

This guard is intended to protect the EMA arithmetic from unrealistic or adversarial inputs. However, because it is enforced as a hard revert in the state-changing split update path, it can prevent legitimate updates.

If the real market ratio ever exceeds this fixed 100x threshold—for example, following a significant change to `r-target`—the split calculator becomes unusable. In that scenario, the DAO/protocol keeper cannot advance the EMA or update `rewards-v7` through the calculator until the ratio falls back below the cap or governance adjusts `r-target`.

As a result, `rewards-v7` may continue using a stale reward split. Rewards could keep being distributed according to an outdated allocation rather than shifting toward the side the calculator would otherwise favor.

Recommendation

Consider removing the hard revert condition `r-current <= r-target * 100` from `compute-and-apply` and replacing it with clamping. If `r-current` exceeds `r-target * 100`, use `r-target * 100` as the effective value for the calculation instead of reverting.

[M-03] New Withdrawal Fee Is Applied Retroactively to Initiated Withdrawals

Location:

- stacking-dao-core-stbtc-v1.clar#L192-L207
- stacking-dao-core-ststxbtc-v1.clar#L167-L190
- stacking-dao-core-stx-v1.clar#L187-L201

Description In `stacking-dao-core`, users can deposit funds and later withdraw them via one of two paths: an instant withdrawal, which is subject to `withdraw-idle-fee`, or a delayed withdrawal, which is subject to `withdraw-fee` and is expected to be lower.

However, when a user initiates a delayed withdrawal, the final fee they will pay is not known at initiation time because the fee percentage is not committed to (or recorded) when the withdrawal is created.

As a result, users may be charged a higher-than-expected fee.

For example:

- Alice deposits into `stacking-DAO-core-stSTXBTC-v1` while the delayed withdrawal fee is `0%`.
- Alice initiates a delayed withdrawal expecting a `0%` fee.
- The DAO updates the delayed withdrawal fee to `1%`.
- After the cooldown period, Alice finalizes her withdrawal and pays the new `1%` fee.

Alice therefore cannot reliably predict her final withdrawal fee or enforce an upper bound on it.

Recommendation

Bind each delayed withdrawal to the `withdraw-fee` value configured at the time the withdrawal is initiated. This can be implemented by storing the applicable fee percentage in the withdrawal NFT.

Additionally, consider enforcing that all withdrawal fee setters must set fees below `100%` to prevent misconfiguration from blocking withdrawals (e.g., due to underflow).

[M-04] SIP-010 Compliance Issues

Location:

- sdao-token.clar
- stbtc-token.clar
- ststx-token.clar
- ststxbtc-token.clar

Description
The four PoX-5 fungible token contracts implement the SIP-010 trait interface, but they do not fully adhere to SIP-010 behavioral requirements. SIP-010 specifies the seven required functions, transfer error-code conventions, memo printing behavior, token URI semantics, and recommends using native FT primitives.

The following compliance issues were identified:

1. All four tokens print the optional memo value directly:

```
(print memo)
```

SIP-010 expects the memo to be printed only when present and to be unwrapped before printing. The current behavior prints `none` when no memo is provided and prints `(some 0x...)` (rather than the raw memo buffer) when a memo is present.

2. All four tokens use custom "unauthorized" errors instead of SIP-010's transfer error-code convention. For example:

```
(define-constant ERR_NOT_AUTHORIZED (err u14001))
```

The standard transfer-style pattern uses `u4` when `sender != tx-sender`. These contracts instead return token-specific error codes such as `u10001`, `u11001`, `u13001`, and `u14001`.

3. All four tokens return `(some u"")` as the default token URI:

```
(define-data-var token-uri (string-utf8 256) u"")
  (define-read-only (get-token-uri) (ok (some (var-get token-uri))))
```

SIP-010 defines `get-token-uri` as an optional metadata URI. An empty string is not a valid URI; until a valid URI is configured, the function should return `none`.

Recommendation

Update all four token contracts to print memos in the SIP-010-compliant form:

```
(match memo to-print (print to-print) 0x)
```

Use the standard transfer authorization error code:

```
(define-constant ERR_NOT_AUTHORIZED (err u4))
```

Store the token URI as an optional value:

```
(define-data-var token-uri (optional (string-utf8 256)) none)
  (define-read-only (get-token-uri) (ok (var-get token-uri)))
```

[M-05] SIP-009 Compliance Issues

Location:

- withdraw-nft-template-v1.clar

Description `withdraw-nft-template-v1` is deployed as:

- `stbtc-withdraw-nft`
- `ststx-withdraw-nft`
- `ststxbtc-withdraw-nft`

The template implements a local mirror of the SIP-009 trait; however, it does not fully align with the official SIP-009 NFT standard.

The following compliance issues were identified:

1. The local SIP-009 trait uses an incorrect URI string type.

Official SIP-009 defines:

```
(get-token-uri (uint) (response (optional (string-ascii 256)) uint))
```

The local trait uses `string-utf8` instead:

```
(get-token-uri (uint) (response (optional (string-utf8 256)) uint))
```

As a result, `withdraw-nft-template-v1` satisfies the local trait but would not satisfy the official SIP-009 trait exactly.

2. `get-token-uri` returns `some` for nonexistent or burned NFTs.

SIP-009 requires `get-token-uri` to return `(ok none)` if the NFT does not exist, or if the contract does not maintain metadata for that token.

The current implementation ignores `token-id` and always returns:

```
(ok (some (var-get token-uri)))
```

This causes `some` to be returned even for token IDs that were never minted or have already been burned.

3. The default token URI is not a valid URI.

The URI is initialized to an empty UTF-8 string:

```
(define-data-var token-uri (string-utf8 256) u"")
```

Before configuration, `get-token-uri` returns `(ok (some u""))`. Under SIP-009, `some` should only be returned when a valid URI exists; otherwise, the function should return `none`.

Recommendation

Update the local SIP-009 trait mirror and template to use `string-ascii` for token URIs:

```
(get-token-uri (uint) (response (optional (string-ascii 256)) uint))
```

Store the URI as optional metadata:

```
(define-data-var token-uri (optional (string-ascii 256)) none)
```

Update `get-token-uri` to return `none` for nonexistent or burned NFTs:

```
(define-read-only (get-token-uri (token-id uint))
    (if (is-some (nft-get-owner? withdraw-nft token-id))
      (ok (var-get token-uri))
      (ok none)
    )
  )
```

Set `token-uri` to `(some "...")` only after a valid metadata URI has been configured.

[M-06] DAO Setters Authorizing `tx-sender` Enable Phished Admin Calls

Location:

- dao.clar#L73
- dao.clar#L82
- dao.clar#L91

Description
The DAO admin setter functions authenticate using the original transaction signer:

```
(try! (check-is-admin tx-sender))
```

This pattern is unsafe for privileged actions because `tx-sender` remains the original signer across nested contract calls. If an admin is tricked into calling an untrusted contract, that contract can then invoke the DAO setter functions while `tx-sender` is still the admin principal.

The affected setters span all DAO authorization types, including:

- `set-contracts-enabled`
- `set-contract-active`
- `set-admin`

A malicious intermediary contract could abuse the admin's `tx-sender` authority to register itself (or another attacker-controlled principal) as an active protocol contract or as an admin. Once registered as a protocol contract, it can call protocol-gated functions throughout the system, including reserve escape-valve functions that can move idle funds.

Recommendation
Authorize DAO setter functions using `contract-caller` instead of `tx-sender`:

```
(try! (check-is-admin contract-caller))
```

[M-07] Incorrect Rounding Direction in stBTC and stSTX Deposits

Location:

- data-stbtc-v1.clar#L18
- data-stx-v1.clar#L21

Description

`data-stbtc-v1::get-sbtc-per-stbtc` and `data-stx-v1::get-stx-per-ststx` both return reserve exchange ratios that are rounded down.

The stBTC and stSTX deposit functions then use these rounded-down ratios as denominators:

```
stbtc-amount = sbtc-amount * DENOMINATOR_8 / ratio
ststx-amount = stx-amount * DENOMINATOR_6 / ratio
```

When the exact reserve ratio is not an integer at the token's precision, rounding down produces a value that is slightly smaller than the true ratio. Using this smaller value as the denominator can result in minting slightly more stBTC or stSTX than would be permitted under exact reserve/supply accounting.

As a result, sBTC and STX deposits may slightly overmint shares.

Recommendation

Introduce a rounding-direction parameter to `get-sbtc-per-stbtc` and `get-stx-per-ststx`. When the ratio is used as a denominator in `deposit`, round the ratio up. When the ratio is used as a multiplier in `init-withdraw` and `withdraw-idle`, round the ratio down. For example:

```
(get-sbtc-per-stbtc round-down)
  (get-stx-per-ststx round-down)
```

Then:

- `deposit`: call with `round-down = false` (round up).
- `init-withdraw` / `withdraw-idle`: call with `round-down = true` (round down).

Alternatively, avoid computing the ratio and instead use total supply and total backing directly. For example:

```
deposit_shares = deposit_amount * total_supply / total_backing
  withdraw_assets = shares * total_backing / total_supply
```

[M-08] Native Signer Manager Access Control Can Be Bypassed

Location:

- native-pool-signer-manager.clar#L39

Description Users delegate STX via `native-pool-v1::delegate`, which forwards funds to `pox-5` for staking through the native signer manager.

During this flow, the user is added to the `registered` map. This map is intended to ensure that only users who delegated through `delegate` can stake via the native signer manager.

However, this restriction can be bypassed because users can interact with the `pox-5` contract directly. For example:

- Alice delegates STX through the native pool and is added to the `registered` map.
- She later unstakes her STX directly via the `pox-5` contract.
- She then stakes again directly through `pox-5`. Because her prior registration remains in `registered`, the stake is still treated as valid.

As a result, users who are no longer actively delegating through the native pool may still be able to stake via the native signer manager.

Recommendation To ensure that only users actively delegating can pass stake validation, implement an "in-progress delegation" flag:

- In `native-pool-v1`, set a flag at the start of `delegate` to indicate an active delegation flow.
- To avoid potential reentrancy issues, key the flag by both the user principal and the signer manager principal.
- Clear the flag at the end of `delegate` once the delegation completes.
- In `validate-stake!`, require that this flag is set for the corresponding `staker` and `current-contract` keys in `native-pool-v1`.

[M-09] stSTX Scheduled Withdrawals Can Drain STX Earmarked for stSTXbtc

Location:

- stx-reserve.clar#L70

Description

`stx-reserve` tracks STX earmarked 1:1 for stSTXbtc principal via `stx-for-ststxbtc`. While most stSTX-related flows preserve this earmark, scheduled withdrawal payouts can bypass it.

The vulnerable path is:

```
(define-public (request-stx-for-withdrawal (requested-stx uint) ...)
    (begin
      ...
      (var-set stx-for-withdrawals (-
        (var-get stx-for-withdrawals) requested-stx))
      (try! (stx-transfer? requested-stx current-contract receiver))
```

Unlike `pay-stx-from-idle` and `get-stx`, this function does not ensure that the reserve's remaining STX balance stays **greater than or equal to** `stx-for-ststxbtc` when the outflow is triggered by **stSTX** (rather than stSTXbtc).

Example scenario:

- Alice deposits 1 STX into stSTXbtc.
- The reserve holds 1 STX and records `stx-for-ststxbtc = 1`.
- Bob deposits 1 STX into stSTX.
- The protocol deploys the non-earmarked stSTX backing via `request-stx-to-stack`, leaving only Alice's stSTXbtc-earmarked STX liquid.
- Bob initiates a scheduled stSTX withdrawal.
- After the cooldown, Bob finalizes the withdrawal.
- `request-stx-for-withdrawal` transfers the only liquid 1 STX to Bob.
- The reserve balance becomes `0` while still recording `stx-for-ststxbtc = 1`.
- Alice's stSTXbtc instant withdrawal fails with `ERR_INSUFFICIENT_IDLE` until additional liquid STX accumulates.

This creates a fund-locking condition for stSTXbtc holders whenever stSTX scheduled withdrawals are paid out while the only liquid STX is stSTXbtc-earmarked backing.

Recommendation
Scheduled withdrawal payouts must not consume STX earmarked for stSTXbtc unless the payout is explicitly releasing stSTXbtc principal. For stSTX withdrawals, enforce that the remaining STX balance after the transfer is sufficient to cover `stx-for-ststxbtc` (i.e., the idle stSTXbtc-backed amount).

[M-10] stSTXbtc Withdrawal Fees Accrue to stSTX Holders

Location:

- stacking-dao-core-ststxbtc-v1.clar

Description The stSTXbtc product mints stSTXbtc 1:1 and distributes yield via the stSTXbtc sBTC reward tracker. However, stSTXbtc withdrawal fees are retained in the shared STX reserve as unearmarked STX.

For instant stSTXbtc withdrawals, `withdraw-idle` burns the full stSTXbtc amount, releases the full stSTXbtc earmark, transfers only the net STX amount to the user, and leaves the fee in `stx-reserve`. For scheduled stSTXbtc withdrawals, `withdraw` releases the fee portion from `stx-for-withdrawals` and transfers only the net STX amount to the user.

In both cases, the retained STX fee is no longer counted as stSTXbtc backing. Because `data-stx-v1::get-stx-per-ststx` subtracts only the remaining stSTXbtc earmark and withdrawal reservations from total STX backing, the retained fee immediately increases the stSTX exchange ratio. As a result, the fee paid by an exiting stSTXbtc holder benefits stSTX holders rather than remaining stSTXbtc holders.

This constitutes a cross-product value transfer unless the protocol explicitly intends stSTXbtc exit fees to subsidize stSTX.

Recommendation Route stSTXbtc withdrawal fees explicitly rather than leaving them as unearmarked STX in the shared reserve. For example, fees could be sent to a protocol treasury, distributed/streamed to stSTXbtc holders via an intended mechanism, or clearly documented as an intentional subsidy to stSTX holders.

8.4. Low Findings

[L-01] Full Boost Collapses Positive TVL to a Discontinuous 100% stBTC Reward Split

Location:

- reward-split-calculator-v1.clar#L106-L107

Description `reward-split-calculator-v1` permits `alpha-max-bps` up to 10,000. This value caps the moving-average-based boost, meaning a 10,000 bps cap corresponds to a maximum boost of ±100%. When the boost reaches the maximum positive value, the opposite side of the split receives a `BPS - boost = 0%` multiplier.

For example, at maximum positive boost:

```
boost = 10000
b-stbtc = 10000 + boost = 20000
b-side  = 10000 - boost = 0
```

If, at the same time, `t-stbtc` is 0 while `t-ststxbtc` and/or `t-ststx` is non-zero, all weighted values evaluate to zero:

```
w-stbtc    = 20000 * 0 = 0
w-ststxbtc = 0 * t-ststxbtc = 0
w-ststx    = 0 * t-ststx = 0
w-sum      = 0
```

`preview` handles `w-sum == 0` by returning:

```
ststxbtc-bps = 0
ststx-bps    = 0
```

However, `rewards-v7` treats stBTC as the implied remainder. As a result, the effective split becomes 100% to the stBTC side.

This creates an unexpected discontinuity: the system can route 100% of rewards to stBTC even when stBTC has zero TVL and all TVL is in the STX-side products. This misallocates rewards away from the products that actually have TVL.

Recommendation

Explicitly handle "positive TVL but zero weight" states instead of allowing the `w-sum == 0` fallback to imply a 100% stBTC allocation. If `w-sum == 0` while total TVL is positive, fall back to an unboosted, TVL-based split among the products with positive TVL. In the maximum positive-boost case where `t-stbtc == 0` and STX-side TVL is non-zero, the full 100% should be split between `ststxbtc` and `ststx` proportionally to their TVLs, rather than being implicitly assigned to stBTC.

[L-02] DAO Can Brick Itself

Location:

- dao.clar#L91

Description The DAO allows any current admin to update admin status via `set-admin`:

```
(define-public (set-admin (address principal) (active bool))
    (begin
      (try! (check-is-admin tx-sender))
      (map-set admins { address: address } { active: active })
      ...
    )
  )
```

However, there is no invariant preventing the last remaining admin from deactivating itself (or deactivating all other admins). If all admins are set to inactive, no principal can call `set-admin`, `set-contract-active`, or `set-contracts-enabled`, since each requires `check-is-admin`. This would permanently brick DAO governance.

This scenario can also occur unintentionally during admin rotation. For example, if an admin-add transaction and an admin-remove transaction are submitted separately and mined in the wrong order, the removal may execute first and leave the DAO with no active admin, causing the subsequent add transaction to fail.

Recommendation Enforce an admin liveness invariant in `set-admin`. When removing an admin, either:

1. Track an `admin-count` and require it to remain at least 1 after any removal, ensuring governance always retains an active admin, or
2. Make the original deployer admin non-removable.

The first approach is preferable to fully support admin rotation.

[L-03] Reward Tracker Exclusion Creates Phantom Supply and Strands Rewards

Location:

- reward-tracker-template-v1.clar#L53

Description The `reward-tracker-template-v1` contract includes an exclusion feature that allows a DAO-controlled set of principals to be fully excluded from rewards.

However, the current implementation does not handle all exclusion-related edge cases.

For example, if the DAO excludes a principal that already has a registered balance, the exclusion is not reconciled with the tracker's accounting. Until the next `refresh-wallet`, the principal can still accrue and claim rewards because `claim-pending-rewards` does not consult the exclusion map. When `refresh-wallet` is eventually called, the principal's tracked amount is overwritten to `0`, but the previous amount is not subtracted from `total-supply`. This leaves phantom supply in the tracker, causing future rewards to be divided across an inflated supply and permanently stranding a portion of rewards as unclaimable.

The documented intent of this feature is to support only the withdrawal escrow principal in the `ststxbtc-tracking-v1` contract. The other instance, `native-pool-tracking-v1`, is expected to have no excluded principals.

Recommendation Because this feature is intended to be used only for the withdrawal escrow, the simplest mitigation is to enforce the following constraints:

- Principals may only be added to the `is-excluded` map and must not be removed.
- A principal may only be added to the `is-excluded` map when its current registered amount is `0`.

[L-04] Native Pool Cannot Enforce DAO-Gated Unstake

Location:

- native-pool-v1.clar#L60

Description `native-pool-v1::undelegate` calls `dao::check-is-enabled` before invoking `pox-5::unstake`.

However, native-pool staking is non-custodial: the user is the actual PoX-5 staker, and their STX is locked by PoX-5 rather than being held by `native-pool-v1`.

As a result, a user can bypass `native-pool-v1` entirely and call `pox-5::unstake` directly via the native signer manager.

Recommendation In the current design, the DAO-enabled check in `undelegate` cannot meaningfully enforce an `unstake` restriction, because `unstake` is directly accessible and the native signer manager is not notified when a staker exits. PoX-5 also does not provide a hook mechanism to enforce such restrictions.

Consider removing the DAO check from `undelegate` entirely.

After this change—and following the removal of `native-pool-tracking-v1` and the update to the native signer manager's stake access—the `is-registered` check becomes redundant and can also be removed. The `undelegate` function can only be called successfully by the native signer manager, whose registration process already ensures the staker originally delegated via `delegate`.

As a result, `undelegate` remains a thin wrapper around `pox-5::unstake`, serving as the counterpart to `delegate`.

[L-05] Lack of Timelock on New DAO Admins Allows Instant Drain on Principal Compromise

Location:

- dao.clar

Description The DAO contract allows any active admin to immediately enable a new DAO admin via `dao::set-admin` or activate a new protocol contract via `dao::set-contract-active`.

This makes the compromise of a single active DAO admin principal immediately catastrophic. A compromised admin can activate malicious protocol contracts within the same block. Once a malicious contract is marked active, it can pass `dao::check-is-protocol`, which is used throughout the system to gate privileged actions such as unlimited token minting/burning, reserve accounting, reserve withdrawals, reward configuration, signer-manager admin assignment, and other protocol-level operations.

As a result, an attacker who compromises a single DAO admin key can rapidly drain protocol funds and cause significant damage before users have time to react.

Recommendation Consider adding a timelock for adding new trusted principals. Specifically, `dao::set-admin(address, true)` and `dao::set-contract-active(address, true)` should be converted into two-step operations:

1. `queue-admin-addition` / `queue-contract-addition` should be callable only by an existing DAO admin and should store the target principal along with an ETA block height at which the admin/contract becomes active.
2. `execute-admin-addition` / `execute-contract-addition` should be callable only after `stacks-block-height >= eta` and should activate the queued admin or protocol contract.

Revocations should remain immediate.

[L-06] sDAO Staking Global Pause Blocks Exits and Reward Claims

Location:

- sdao-staking-sbtc-v1.clar

Description `sdao-staking-sbtc-v1` relies on `dao::check-is-enabled` to gate both entry and exit flows.

This single global check is applied to `stake`, `unstake`, and `claim-pending-rewards`. Consequently, disabling the contract globally to prevent new sDAO staking also prevents users from withdrawing previously staked sDAO and claiming already-accrued sBTC rewards.

This behavior makes the global pause unsuitable as a migration control for `sdao-staking-sbtc-v1`. During a migration, new staking may need to be redirected to an upgraded contract, while existing users must still be able to exit and claim rewards from the legacy contract that maintains their staking state and reward accounting.

Recommendation Introduce an entry-only pause for staking (e.g., a `shutdown-stake` flag) and use it for migrations. Keep `unstake` and `claim-pending-rewards` available, or gate them behind separate flags such as `shutdown-unstake` and `shutdown-claim-pending-rewards`.

[L-07] Scheduled Withdrawal Cooldown May Mature Before Reserve Liquidity Is Available

Location:

- stacking-dao-core-stbtc-v1.clar#L69
- stacking-dao-core-stx-v1.clar#L68
- stacking-dao-core-ststxbtc-v1.clar#L56

Description
The scheduled-withdrawal data contracts store a DAO-controlled `withdraw-cooldown-blocks` parameter. When a user initiates a scheduled withdrawal, the core contract records an `unlock-burn-height` equal to the current burn block height plus this cooldown.

However, this cooldown only determines when the withdrawal NFT becomes eligible for finalization; it does not ensure the reserve will have sufficient idle assets to satisfy the withdrawal at that time.

Backing assets may be deployed into PoX-5 staking or bond positions. STX-only stakes can only be unstaked at a future reward-cycle boundary, and unstakes are blocked during the prepare phase. As a result, STX-only collateral may require up to a full reward cycle plus the prepare phase to become liquid. Additionally, bonded sBTC—and the STX collateral used for sBTC bonds—remains locked for the full bond term (12 reward cycles).

Consequently, if `withdraw-cooldown-blocks` is configured shorter than the applicable liquidity unwind period, users may hold withdrawal NFTs that have matured (i.e., pass the cooldown check) but still cannot be paid because the reserve lacks idle liquidity. This creates a mismatch between the user-visible withdrawal maturity time and the protocol's actual liquidity availability.

Recommendation
Enforce a minimum value for `withdraw-cooldown-blocks` that aligns with the maximum expected liquidity unwind time for the product, or explicitly document that the cooldown only governs NFT maturity and does not guarantee immediate liquidity.

[L-08] Withdrawal Fees Cause Immediate Price Ratio Increases

Location:

- stacking-dao-core-stbtc-v1.clar
- stacking-dao-core-ststxbtc-v1.clar
- stacking-dao-core-stx-v1.clar

Description Withdrawal fees are currently retained directly in the backing reserve rather than being streamed over time or segregated.

For instant withdrawals, users burn their full share balance but receive only the net asset amount. The fee portion remains in the reserve, increasing the exchange ratio for remaining holders within the same transaction.

For delayed withdrawals, the full gross withdrawal amount is reserved at initiation. Upon finalization, the user receives only the net amount, while the fee portion is released from the reservation and returned to the active reserve. This causes a predictable increase in the exchange ratio at finalization.

In both cases, a user can deposit immediately before a large, fee-generating withdrawal, allowing newly minted shares to benefit from the resulting increase in the price ratio.

This strategy becomes economically viable when the withdrawal fees captured exceed the depositor's own withdrawal costs and any risks associated with liquidity delays.

Recommendation Consider redirecting withdrawal fees to a dedicated contract that streams them into the product's price ratio gradually over time.

Alternatively, since participants are already disincentivized by their own withdrawal fees and exposure to liquidity risk, this behavior can be explicitly acknowledged and documented.

8.5. QA Findings

[QA-01] Deprecated Contracts Can Be Removed

Location:

- Clarinet.toml#L261
- ststxbtc-data-v1.clar

Description Reward accounting for the native pool is now managed by the `pox-5` and `native-signer-manager-v1` contracts. As a result, `native-pool-tracking-v1` is obsolete. Any continued interaction with this contract unnecessarily increases user transaction costs, and the balance data it exposes is no longer reliable.

Additionally, `ststxbtc-data-v1.clar` is a legacy predecessor to `withdraw-data-template-v1.clar` and is not included in the latest deployment configuration.

Recommendation Remove any remaining references to, and interactions with, the `native-pool-tracking-v1` contract. Also remove the obsolete `ststxbtc-data-v1.clar` implementation.