ClarityAlliance

Zest Protocol's DEX Aggregator Security Review

Conducted by:
Kristian Apostolov

July 17th, 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 treasury and adapter Clarity contracts, together with the Rust shard generator used by Zest Protocol's Stacks DEX aggregator. The generated output of the shard generator, backend systems, and other off-chain components were not comprehensively reviewed unless referenced for context.

4. About Zest Protocol

What is Zest Protocol?

Zest Protocol is the leading Bitcoin lending protocol. BTC holders earn yield on their Bitcoin and borrow stablecoins against it.

Zest Protocol is the largest lending protocol on Bitcoin L2s, with $100M+ peak TVL and over two years of mainnet history. Built by one of the longest-running team in Bitcoin DeFi, building on Bitcoin since 2020.

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: `zest-agg` (Zest Protocol's Stacks DEX aggregator)

Scope covers the treasury, the on-chain DEX adapter contracts, and the Rust shard generator.

Adapters

`contracts/src/`

- alex-adapter.clar
- arkadiko-adapter.clar
- bitflow-adapter.clar
- dlmm-adapter.clar
- stableswap-adapter.clar
- stacking-dao-adapter.clar
- stacking-dao-receipt-adapter.clar
- velar-adapter.clar
- velar-stableswap-adapter.clar

Shard generator

`crates/sdex-shard-gen/`

- src/lib.rs
- src/main.rs

Treasury

`contracts/src/`

- treasury.clar

Initial Commit Reviewed

`abc43de70a69257bb4c638475d831541c93d7922`

Final Commit Reviewed

`67ddafc085f18a929f910254d770a4c03ac400f1`

7. Executive Summary

Over the course of the security review, Kristian Apostolov engaged with Zest to review Zest Protocol. In this period of time a total of **9** issues were uncovered.

Protocol Summary

Protocol Name	Zest Protocol
Repository	zest-agg
Report Date	July 17th, 2026

Findings Count

Severity	Amount
Medium	5
Low	4
Total Findings	**9**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Single-Step Treasury Ownership Can Forfeit All Fees	Medium	Resolved
[M-02]	A One-Unit Donation Bricks the DLMM Adapter	Medium	Resolved
[M-03]	ALEX Adapter Scales Swaps Using Token-Reported Decimals	Medium	Resolved
[M-04]	Sponsored Stacking DAO Leg Lacks a Replay Guard	Medium	Resolved
[M-05]	Incomplete Deny-Mode Post-Condition Set Aborts Valid Swaps	Medium	Resolved
[L-01]	Simnet mock-wstx Principals Remain in the Production Treasury	Low	Resolved
[L-02]	Generic Adapters Omit the Input-Consumption Assert	Low	Resolved
[L-03]	Router Custody Core Lacks Exact-Delivery Assertions	Low	Resolved
[L-04]	Router Output Post-Condition Omits the Slippage Floor	Low	Resolved

8. Findings

8.1. Medium Findings

[M-01] Single-Step Treasury Ownership Can Forfeit All Fees

Location:
`contracts/src/treasury.clar#L47-L62`, consumers `sweep-ft#L20-L33` and `sweep-stx#L34-L42`

Description
The treasury accrues all skimmed protocol fees, and the only withdrawal methods, `sweep-ft` and `sweep-stx`, are gated by a single owner via `is-owner`. Ownership is transferred in a single step by directly writing the new owner. As a result, one mistyped or compromised call can transfer sole authority over all accrued and future fees to an arbitrary principal, including one that does not exist. This transfer cannot be reversed because the former owner no longer satisfies `is-owner`.

Recommendation
Implement a two-step ownership transfer. The current owner proposes a candidate, and the candidate must accept from its own key before the change takes effect. The proposal should remain revocable until acceptance so that an accidental transfer cannot become final.

[M-02] A One-Unit Donation Bricks the DLMM Adapter

Location: `contracts/src/dlmm-adapter.clar#L30-L46`

Description The DLMM adapter is stateless and asserts that no input remains after a swap. However, it performs this check against the adapter's absolute token balance, requiring the input-token balance to be exactly zero after the swap completes. Because the adapter is a fixed principal, anyone can transfer tokens to it. As a result, sending a single base unit to the adapter leaves a non-zero balance that subsequent swaps cannot clear, causing every swap routed through the adapter to revert until the dust is removed. This effectively bricks the adapter at the cost of one token unit and provides no benefit to the attacker, making it a pure griefing vector.

Recommendation Track the input-token balance before pulling funds and assert that the post-swap balance has not increased relative to that baseline (i.e., the net remaining input from the swap is zero). This ensures that any prior donation only raises the baseline and does not block future swaps. Avoid asserting an absolute zero balance on a stateless adapter that can be externally funded.

[M-03] ALEX Adapter Scales Swaps Using Token-Reported Decimals

Location: `contracts/src/alex-adapter.clar#L123-L153`

Description ALEX pools trade eight-decimal wrapper tokens, while the adapter pulls the user's canonical token. To bridge this mismatch, the adapter scales the pulled amount before calling the AMM. It derives the scale factor from the input token's `get-decimals` value and applies it to an actual transfer:

```
(let ((dec (try! (contract-call? token-in get-decimals)))
      (mul (if is-stx-in u100 (pow u10 (- u8 dec)))))
  (try! (contract-call? token-in transfer
    (* amount-in mul) tx-sender current-contract none))
```

Because the multiplier is controlled by the token contract, a token that misreports its decimals—or a wrapper/custody pair that does not correspond—can cause the adapter to mis-scale a swap leg that moves funds. While measuring the output based on the actual balance change and enforcing `min-out` bounds limits the loss, it does not prevent the mis-scaling from occurring.

Recommendation Derive the scale factor from a reviewed wrapper-to-custody registry rather than from the token contract. Require the caller's custody principal to match the registry entry, and assert that the pulled input amount is fully consumed.

[M-04] Sponsored Stacking DAO Leg Lacks a Replay Guard

Location: `contracts/src/stacking-dao-receipt-adapter.clar#L38-L87`

Description The Stacking DAO source hop executes via the sponsored path, where the sponsor fronts transaction fees in exchange for a verified output skim. However, there is no mechanism that binds a sponsored execution to a single-use receipt. The adapter relies solely on `tx-sender` balance deltas, which allows a sponsored receipt to be replayed or forged, enabling the same leg to be executed multiple times under the production sponsor's account.

Recommendation Introduce a map keyed by the transaction sponsor and a receipt ID. On the sponsored path, require a fresh, non-zero receipt ID, and record it only after the amount, delta, and floor checks succeed. This ensures a receipt is created only when the leg fully commits:

```
(define-private (require-receipt (receipt-id uint))
  (match tx-sponsor?
    sponsor (begin
      (asserts! (> receipt-id u0) ERR_RECEIPT)
      (asserts! (not (get-receipt sponsor receipt-id)) ERR_RECEIPT_USED)
      (ok true))
    (begin (asserts! (is-eq receipt-id u0) ERR_RECEIPT) (ok true))))
```

Keying by the actual sponsor prevents an unrelated caller from forging a receipt on behalf of the production sponsor.

[M-05] Incomplete Deny-Mode Post-Condition Set Aborts Valid Swaps

Location: `crates/sdex-calldata/src/lib.rs` `route_asset_footprint`

Description The router transaction is constructed in deny post-condition mode, meaning the node aborts if any contract transfers an asset that is not covered by a post-condition. The builder currently specifies only the user's input, but a swap transfers assets across multiple contracts, including the router shard, adapters, pool cores, the treasury, and wSTX facades. As a result, multi-hop routes can fail due to a post-condition violation even when the swap is otherwise valid, preventing affected routes from executing.

Recommendation Generate post-conditions for every contract that may transfer assets, using the same route legs that are used to build the calldata. Pin the user's input to an exact amount, and allow any amount for internal legs where the transferred amounts cannot be known ahead of time. The `settle-and-floor` min-out assertion should remain the primary value guarantee, so permissive internal post-conditions do not reduce slippage protection.

8.2. Low Findings

[L-01] Simnet mock-wstx Principals Remain in the Production Treasury

Location: `contracts/src/treasury.clar#L9-L19`

Description The production treasury's `is-wstx` function includes three simnet-only mock principals alongside the real facades:

```
(define-private (is-wstx (token principal))
  (or
    (is-eq token .mock-wstx)
    (is-eq token .mock-wstx-2)
    (is-eq token .mock-wstx-8)
    ...
```

On mainnet, these principals resolve to undeployed deployer contracts and are only referenced in `is-eq` comparisons against a token principal. As a result, no real token can match them. They do not gate funds, do not block publishing, and `sweep-ft` remains owner-only regardless. These entries are simnet artifacts left in a production contract solely to satisfy the `wstx_set_consistency` check, which currently ties the treasury's wSTX set to the simnet project's set.

Recommendation

Remove the three mock principals and update `wstx_set_consistency` to compare against the production wSTX set so the check passes without them.

[L-02] Generic Adapters Omit the Input-Consumption Assert

Location: `contracts/src/bitflow-adapter.clar#L32-L46`, and the stableswap, velar, velar-stableswap, and arkadiko adapters

Description These five adapters pull the input, grant the venue a `with-ft` ceiling, and measure only the output delta, without asserting that the venue consumes the entire input. A `with-ft` grant is a ceiling rather than an exact spend; therefore, if a venue takes less than the full amount, some input can remain stranded in the adapter, which has no sweep mechanism. The DLMM and ALEX adapters assert that the input is fully consumed; these five do not. Any stranded amount would be the swapper's own dust and cannot be extracted, since output is measured from a pre-pull baseline. Additionally, no currently integrated venue under-consumes, so the practical risk is low.

Recommendation Assert that the input is fully consumed in all five adapters by comparing the post-swap input balance against a baseline taken before the pull, so that any future under-consuming venue fails closed.

[L-03] Router Custody Core Lacks Exact-Delivery Assertions

Location:
`contracts/src/generated/shards/zr-a-0.clar`, generator `crates/sdex-shard-gen/src/lib.rs` `core()`

Description
The shard custody core is generated and shared byte-for-byte across all shards. It pulls inputs and pays outputs without asserting that balances change by exactly the intended amounts. As a result, fee-on-transfer or rebasing tokens—or a pool that transfers a different amount than requested—can desynchronize the router's tracked custody from actual on-chain balances.

Recommendation
Introduce a helper for inbound transfers that asserts the router's balance increases and the sender's balance decreases by exactly the specified amount. Similarly, route outbound transfers through a helper that asserts the recipient is credited and the router is debited by exactly the specified amount, in addition to the existing STX-float invariant. Regenerate all shards via the generator to ensure these assertions remain identical across shards.

[L-04] Router Output Post-Condition Omits the Slippage Floor

Location: `crates/sdex-calldata`, `crates/sdex-api` build path

Description The router output post-condition currently checks only that the received amount is greater than or equal to zero. As a result, the slippage floor is enforced solely by the on-chain `min_out` assertion and is not reflected in the post-condition presented to the signing wallet. The wallet therefore displays the swap as receiving at least zero, preventing the user from seeing the actual minimum amount they are guaranteed to receive.

Recommendation Set the output post-condition to the same `min_out` value enforced on-chain, and place it before the coverage legs. This ensures the wallet displays "send exactly the input" and "receive at least the floor." Since execution is already floored on-chain, this strengthens the wallet-facing guarantee without changing runtime behavior.