ClarityAlliance

ALEX LAB SECURITY REVIEW

Conducted by:
KRISTIAN APOSTOLOV, ALIN BARBATEI (ABA)

MAY 16TH, 2025

CONTENTS

ClarityAlliance
Security Review

ALEX Lab

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

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

3. Introduction

A time-boxed security review of the ALEX Lab, where Clarity Alliance reviewed the scope and provided insights on improving the protocol.

4. About ALEX

What is ALEX?

"ALEX is building the finance layer on Bitcoin. The ALEX DEX is the largest on Bitcoin layers (Stacks Chain) fully integrated with XLink, our cross-chain bridge aggregating liquidity across L2s and multi-chain, with LISA as our liquid staking platform.

We're creating a seamless user experience, enabling one-click trading and asset transfer across blockchains that abstract away wallet and network complexity. All roads lead to Bitcoin, and all roads on Bitcoin meet on ALEX.

There is close to $1T of capital asleep in Bitcoin wallets, this is an ocean of money that ALEX seeks to awaken. ALEX unlocks the potential of Bitcoin by taking the ultimate store of value and building on top of it the first truly permissionless, trustless and decentralized financial service for the people.

ALEX offers a suite of DeFi opportunities that includes:

- Discover and participate in the IDO rounds of emerging projects through the Launchpad
- AMM DEX with deep liquidity
- Earn exciting returns through providing liquidity, $ALEX staking, and yield farming
- Cross-chain bridging through XLink from Bitcoin L1, to L2s and EVM chains.
- Liquid token staking through LISA.
- Advanced order-book DEX allows limited orders and market orders.

Just as Bitcoin is the "gold standard" of crypto, ALEX will become gold standard of DeFi."

CONTENTS

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

CONTENTS

ClarityAlliance
Security Review

ALEX Lab

6. Security Assessment Summary

Scope:

The following contracts were in the scope of the security review:

- `contracts/amm-liquidity-token-v3.clar`
- `contracts/amm-pool-v3.clar`
- `contracts/amm-pool-v3-helper.clar`

Initial Commit Reviewed:
180b7bdd1d6608928ec9590155add68f5dc68284

Final Commit After Remediations:
921d997a9f3bc735ca8d94742d101f21a6db1205

CONTENTS

7. Executive Summary

Over the course of the security review, Kristian Apostolov, Alin Barbatei (ABA) engaged with - to review ALEX. In this period of time a total of **21** issues were uncovered.

Protocol Summary

Protocol Name	ALEX
Date	May 16th, 2025

Findings Count

Severity	Amount
Medium	3
Low	8
QA	10
Total Findings	**21**

CONTENTS

ClarityAlliance
Security Review

ALEX Lab

Summary of Findings

ID	Title	Severity	Status
[M-01]	V3 Liquidity Token Is Not SIP-13 Compliant	Medium	Resolved
[M-02]	Direct Burning of LP Tokens Results in Unbacked Pool Total Supply	Medium	Resolved
[M-03]	Missing Minimum Amount Checks When Removing Liquidity	Medium	Acknowledged
[L-01]	Inability to Set Long LP Token Symbol Name	Low	Resolved
[L-02]	Duplicated But Reversed Pools Can Be Created	Low	Resolved
[L-03]	Swapping on Empty Tick Intervals Reverts with Panic	Low	Resolved
[L-04]	Ambiguous Swap Revert on Transfers Due to Unchecked Zero Transfer Amounts	Low	Resolved
[L-05]	Inadequate Tick Range Checks	Low	Resolved
[L-06]	Some Swap Functions Are Missing Slippage Checks	Low	Acknowledged
[L-07]	AMM Operations Lack Deadline	Low	Acknowledged
[L-08]	Unsorted Tick Array Results in Suboptimal Swap Prices	Low	Acknowledged
[QA-01]	Token LP Contract Supports but Does Not Implement the SIP-13 Send Many Trait	QA	Resolved
[QA-02]	Reducing LP Position May Burn Dust Liquidity Without Affecting Balances	QA	Acknowledged
[QA-03]	Ambiguous Revert on Zero LP Amount Reduction	QA	Resolved
[QA-04]	Implicit Minimum Token Transfer Amount	QA	Acknowledged
[QA-05]	Absence of Events for Critical Actions	QA	Resolved
[QA-06]	Codebase Naming Improvement	QA	Resolved
[QA-07]	Overlapping or Discontinuous Error Code Ranges	QA	Acknowledged
[QA-08]	Add Preview-Operations Functions	QA	Acknowledged
[QA-09]	Codebase Can Be Slightly Optimized	QA	Resolved
[QA-10]	Include Returned Token Amounts in reduce-position Output	QA	Resolved

CONTENTS

8. Findings

8.1. Medium Findings

[M-01] V3 Liquidity Token Is Not SIP-13 Compliant

Description

The Alex DAMM V3 LP contract, `amm-liquidity-token-v3` , is an implementation of the SIP-13 semi-fungible token standard.

However, the current implementation does not comply with SIP-13 due to the handling and emission of `memo` in relation to the `transfer` and `transfer-memo` functions.

According to the SIP13:Events section, emitted events should:

> be emitted after any built-in token events (such as those emitted by `ft-transfer?`) and before the memo in the case of `transfer-memo` and `transfer-many-memo` .

The specification requires that an `sft-transfer` event, containing a `{type: "sft_transfer", token-id: uint, amount: uint, sender: principal, recipient: principal}` tuple structure, should be emitted when tokens are transferred.

In the current implementation, all transfers, whether using the `memo` version or not, emit the `sft_transfer` event with an additional `memo` entry:

```
(
   print{type:"sft_transfer",
   token-id:token-id,
   amount:amount,
   sender:sender,
   recipient:recipient,
   memo:memo}
)
```

If the simple `transfer` version is used, the `memo` field is empty but still emitted. This violates two specifications in the SIP-13 standard: the `memo` must be emitted after the sft_transfer event, and the `sft_transfer` event structure itself must not contain any `memo` field.

Besides compliance issues, the extra tuple element in the `sft_transfer` event slightly increases execution costs for every transfer.

CONTENTS

Recommendation

Move the current `amm-liquidity-token-v3::transfer-memo` implementation to the `transfer` function while removing the `memo` field from the `sft_transfer` printed tuple. The new `transfer-memo` function should print the memo after executing a normal transfer, as demonstrated in the SIP full example.

ClarityAlliance
Security Review
ALEX Lab

[M-02] Direct Burning of LP Tokens Results in Unbacked Pool Total Supply

Description

When a user adds liquidity to a pool position using the `amm-pool-v3::add-to-position` function, a certain amount of LP tokens are minted via the `amm-liquidity-token-v3::mint` function. These tokens are then tracked in the `pool-supply::total-supply` mapping element.

If a user wants to reduce their position, they can call `reduce-position`, which burns the corresponding amount of LP tokens from the LP contract using `amm-liquidity-token-v3::burn`. This action correctly updates the `pool-supply::total-supply` mapping element with the reduced amount.

However, users are also permitted to directly burn LP tokens by calling the `amm-liquidity-token-v3::burn` function.

When LP tokens are burned directly, the internal accounting of `total-supply` within the pool contract is not updated, leading to the existence of ghost, unredeemable LP tokens. These ghost LP tokens result in other LP holders within the same tick interval receiving less yield than intended. This issue arises because the `total-supply` for the tick interval incorrectly includes the burned LP tokens.

Recommendation

Prevent the direct burning of LP tokens through `amm-liquidity-token-v3::burn`. Instead, implement an equivalent `burn` function within the `amm-pool-v3` pool contract that also updates the internal `pool-supply::total-supply` mapping element.

If the intended behavior is to allow users to lock liquidity, then restrict the `burn` mechanism to protocol-approved addresses only, as users can transfer the LPs to an equivalent, inaccessible burn principal.

CONTENTS

[M-03] Missing Minimum Amount Checks When Removing Liquidity

Description

From a liquidity provider's perspective, the `amm-pool-v3` pool contract functions like any other AMM. Providers deposit tokens, receive LP tokens, and can exchange these LP tokens for their share of the underlying tokens, plus any accumulated fees.

Due to the nature of AMMs and market volatility, the amount of tokens a user receives, while maintaining their percentage of the pool, may not meet the initially desired amount of output tokens.

To address this, slippage protection mechanisms are typically employed.

In the current implementation, both the `amm-pool-v3` pool contract and the `amm-pool-v3-helper` router contract lack slippage protection in any of the liquidity removal wrappers.

This can result in users receiving fewer tokens than expected.

Recommendation

Modify the `amm-pool-v3-helper::reduce-positions-many` function so that the `positions` array includes a minimum `X` and `Y` returned amount within the tuples.

Add the returned amounts of `X` and `Y` tokens to the output tuple of the `amm-pool-v3::reduce-position` function, and then use these amounts to compare against the required minimum in the `reduce-positions-iter` function call.

CONTENTS

ClarityAlliance
Security Review

ALEX Lab

8.2. Low Findings

[L-01] Inability to Set Long LP Token Symbol Name

Description

The `amm-liquidity-token-v3` contract includes a data variable designed to store the symbol of semi-fungible tokens, with a maximum length of 32 characters:

```
(define-data-var token-symbol (string-ascii 32) "amm-liquidity-token-v3")
```

Although the token symbol can be up to 32 characters long, the current symbol setter function, `set-symbol` , restricts the length of the string to a maximum of 10 characters.

```
(define-public  (set-symbol (new-symbol (string-ascii 10)))
        (begin
                (try! (is-dao-or-extension))
                (ok (var-set token-symbol new-symbol))))
```

This limitation hinders any future changes to the symbol that may require more than 10 characters. For instance, the default value of the current symbol exceeds 10 characters.

Recommendation

Update the `amm-liquidity-token-v3::set-symbol` function to permit the `new-symbol` parameter to have a length of up to 32 characters.

CONTENTS

[L-02] Duplicated But Reversed Pools Can Be Created

Description

When a new pool is created using the `amm-pool-v3::create-pool` function, a deduplication check is performed to ensure that no pools with the same X and Y tokens are created:

```
(let (
  ;; ... code ...
  (pool-key { token-x: (contract-of token-x-trait), token-y:
    (contract-of token-y-trait), bin-size: bin-size })
  ;; ... code ...
(asserts! (is-none
  (map-get? pool-id-by-token pool-key)) ERR-POOL-ALREADY-EXISTS)
```

However, there is no check to prevent the creation of a pool with the same two tokens in reverse order (e.g., setting the original X token as Y and the original Y token as X).

Semantically, swapping from X to Y is identical to swapping from Y to X. Thus, allowing the same pair of tokens, with the same bin, to exist in reverse is equivalent to permitting a duplicated pool.

Recommendation

Add a check in `amm-pool-v3::create-pool` to ensure that the reverse token pair is also not present in the `pool-id-by-token` mapping.

CONTENTS

[L-03] Swapping on Empty Tick Intervals Reverts with Panic

Description

When a swap is initiated on an empty tick interval, where no liquidity has been added, the swap results in a division by zero panic error. This issue occurs regardless of whether the swap is from `X` to `Y` or from `Y` to `X`, and whether it is executed through the helper contract or directly via the `amm-pool-v3` contract.

For instance, in the `amm-pool-v3::swap-x-for-y-ioc` function, the division by zero occurs when calculating the intermediary `dy` variable output amount:

```
(dy (- vyy (/ k (+ vxx (- actual-dx fee)))))
```

The problem arises because, in the division `(/ k (+ vxx (- actual-dx fee)))`, if the pool is empty, all values, including `k` and `vxx`, are zero.

Similarly, for the `swap-x-for-y-ioc` function, the panic is triggered when calculating the `dx` output amount variable:

```
(dy (- vxx (/ k (+ vyy (- actual-dy fee)))))
```

Again, the issue occurs because `k`, `vyy`, and the other variables are all zero in the `(+ vyy (- actual-dy fee))` denominator.

If the protocol logic is to treat these cases as pass-throughs, then they need to be resolved. Otherwise, if the revert is intended by the team, it should be identified and resolved earlier in the swaps in a clear manner to facilitate debugging.

Recommendation

If a revert is intended when swapping through empty ticks, then in both the `swap-x-for-y-ioc` and `swap-y-for-x-ioc` functions, ensure that `k`, after calculating the pool virtual balances, is not zero. Example implementation:

```
-    (k (* vxx vyy))
+    (k-unchecked (* vxx vyy))
+    (k (try! (if (> k-unchecked u0) (ok k-unchecked) ERR-POOL-NOT-FOUND)))
```

CONTENTS

If a pass-through is intended in these cases, ensure that the denominators in each division are not zero before attempting the division, or default to zero.

An example implementation for the `swap-x-for-y-ioc` function:

```
-    (dy (- vyy (/ k (+ vxx (- actual-dx fee)))))
+    (dy-unchecked (+ vxx (- actual-dx fee)))
+    (dy (if (> dy-unchecked u0) (- vyy (/ k dy-unchecked)) u0))
```

And an example for the `swap-y-for-x-ioc` function:

```
-    (dx (- vxx (/ k (+ vyy (- actual-dy fee)))))
+    (dx-unchecked (+ vyy (- actual-dy fee)))
+    (dx (if (> dx-unchecked u0) (- vxx (/ k dx-unchecked)) u0))
```

CONTENTS

[L-04] Ambiguous Swap Revert on Transfers Due to Unchecked Zero Transfer Amounts

Description

When executing a swap in the pool contract, the input amount (either `X` or `Y`) to be swapped is calculated based on several mathematical operations and constraints. If this resulting amount is greater than 0, it is assumed that a corresponding non-zero output amount (`Y` or `X`, respectively) has been determined and can be sent to the swapper.

However, there are rare cases where the amount to be swapped is a non-zero value, but the output amount is 0. This means that if the swaps were executed, users would receive 0 output tokens while losing input tokens.

These swaps already revert due to attempting to transfer a 0 amount of tokens from the pool contract, but they revert with the error code `3`, as noted in the SIP-10 transfer non-positive value error.

At a protocol level, there are two considerations regarding this finding:

- If the intended behavior is to skip these cases (perform no token transfers and exit without reverting), then a fix is required.
- If the intended behavior is to revert in these cases, a more specific error is needed to assist external integrators in their debugging.

Recommendation

If the required fix is to allow these cases as a passthrough, then in both swap functions from the `amm-pool-v3` contract, `swap-y-for-x-ioc` and `swap-x-for-y-ioc`, ensure that both `actual-dx` and `actual-dy` are greater than 0 simultaneously.

If a revert is intended, in each case, after verifying that the input amount is different from 0 (`actual-dx` for `swap-y-for-x-ioc` and `actual-dy` for `swap-x-for-y-ioc`), assert that the corresponding output amount is greater than 0 and revert with a specific error if it is not.

[L-05] Inadequate Tick Range Checks

Description

All user-facing actions that accept a tick as input perform a validation to ensure it falls within a valid range. This check is implemented in the `amm-pool-v3::ensure-tick-in-range` function, which verifies that the tick is within the `[-10000, 10000]` range.

```
(define-read-only (ensure-tick-in-range (tick int))
      (if (and (>= tick -10000) (<= tick 10000)) (ok tick) ERR-TICK-OUT-OF-RANGE))
```

However, this check is overly broad and allows values that could cause execution to revert. The tick is used for identification (e.g., via the `get-liquidity-token-id` function) and for calculating virtual balances through the `get-virtual-balances` function.

In the `get-virtual-balances` function, there is a check that causes the function to revert if the tick interval start price is 0, as seen here. This check will revert for a significant number of accepted tick ranges because the function that calculates the start price for the range, `tick-to-price`, defaults to 0 in some cases.

The `tick-to-price` function accepts a `bin` and `tick` argument. Since bins are limited to `[1, 5, 10, 20]` and ticks to `[-10000, 10000]`, there are only 80,004 different input variations. Out of these 80,004, non-zero prices are returned only within the following ranges:

bin	tick min	price-start-min	tick max	price-start max
1	-1851	1	2047	70,121,649,362,215,812
5	-377	1	511	6,725,612,181,217,321,284
10	-193	1	255	3,590,332,865,940,255,062
20	-101	1	127	1,137,675,084,055,324,467

Note that in the spreadsheet, the price validation logic was implemented as it is done in the protocol math tests, with an added 0 price check.

The results indicate that `tick-to-price` can be limited to `[-1851, 2047]`, which will still revert if the bin is not 1 and the tick is outside the limits of other bins.

There are additional limitations at these extreme tick intervals due to the design of the `get-virtual-balances` function.

```
(define-read-only (get-virtual-balances (bin-size uint) (tick int)
  (balance-x uint) (balance-y uint))
        (let (
                              (price (tick-to-price bin-size tick))
                              (ts (+ ONE_8 (* bin-size u1000000)))
                              (t (sqrti (* ONE_8 ts)))
                              (pt (mul-down price t))
                              (x-pty (+ balance-x (mul-down pt balance-y)))
                              (dd (* u2 (mul-down price (- ts t))))
                              (de (+ x-pty (sqrti (+ (* x-pty x-pty) (* dd u2
    (mul-down balance-x balance-y))))))
                              (vy (div-down de dd))
                              (vx (div-down de (* u2 (- t ONE_8))))
                )
                (unwrap-panic (if (> price u0) (
  unwrap-panic
 )
  (ok { vx: vx, vy: vy, price-start: price, price-end: (mul-down price ts
```

The maximum negative value is not reachable, regardless of the bin, due to a divide-by-zero error if `balance-x` is not 0.

If `balance-x` is not 0, then the `price` (returned by `tick-to-price`) must be greater than or equal to `ONE_8 / (ts - t)` to ensure that the `dd` variable (`(dd (* u2 (mul-down price (- ts t))))`) does not become 0, which would cause a divide-by-zero error in the `vy` calculation (`(vy (div-down de dd))`).

If `balance-x` is 0, then `balance-y` can be up to 999999999 (`ONE_8 - 1`), allowing for the maximum negative tick values. This second limit ensures that `x-pty` (`(x-pty (+ balance-x (mul-down pt balance-y)))`) is 0, which makes `de` also 0.

A value of 0 for `de` will trigger the first 0 check in the div-down division when calculating `vy` (`(vy (div-down de dd))`), which in turn skips dividing with the zeroed `dd` .

Considering these factors, the following limitations apply for a non-reverting `get-virtual-balances` call:

tick range	bin	balance-x	balance-y
`[-1851, -1317]`	1	0	`[0, 99999999]`
`[-1381, 2047]`	1	any	any
`[-377, -300]`	5	0	`[0, 99999999]`
`[-301, 511]`	5	any	any
`[-193, -160]`	10	0	`[0, 99999999]`
`[-161, 255]`	10	any	any
`[-101, -86]`	20	0	`[0, 99999999]`
`[-87, 127]`	20	any	any

Observations

Ticks -1318 for bin 1, -301 for bin 5, -161 for bin 10, and -87 for bin 20 are the first tick values where the condition `price >= ONE_8 / (ts - t)` is satisfied.

Within the negative tick intervals reached when `balance-x` is 0 and `balance-y` is less than `99999999`, both the resulting virtual X (`vx`) and virtual Y (`vy`) balances are 0. In other words, a user can initially supply a Y token liquidity up to 0.99999999 full units to any position and have the resulting virtual supplies 0, meaning a slightly imbalanced pool.

Also, users that add liquidity in these specific negative ranges will have their liquidity inaccessible the moment `balance-y` reaches `ONE_8` or `balance-x` becomes non-zero. This is equivalent to dust.

Note, the previously mentioned spreadsheet also recreates the `get-virtual-balances` corner-cases in a sheet with the same name.

Recommendation

Modify the `ensure-tick-in-range` function to take into consideration `bin` value and use the negative tick limits derived by having the `price >= ONE_8 / (ts - t)` condition always fulfilled:

bin	tick range
1	`[-1318, 2047]`
5	`[-301, 511]`
10	`[-161, 255]`
20	`[-87, 127]`

CONTENTS

[L-06] Some Swap Functions Are Missing Slippage Checks

Description

In Automated Market Makers (AMMs), due to normal market volatility, the pool balances may change between the time a user initiates a swap and when it is executed. This can result in the user receiving fewer tokens than expected, a phenomenon known as slippage. To mitigate this, users typically provide a minimum output amount parameter to the swap functions, ensuring that if this minimum is not met, the swap will fail.

The swap operations within the `amm-pool-v3` contract are designed in such a way that incorporating slippage checks for each tick interval is not feasible. However, in the `amm-pool-v3-helper` contract, all swap functions should include a slippage check.

There are five swap functions in the helper contract: `swap-x-for-y-fok`, `swap-y-for-x-fok`, `swap-x-for-y`, `swap-y-for-x`, and `swap-routes`. Among these, only swap-routes has implemented a minimum token out slippage check.

Users utilizing any of the other swap functions may experience significant slippage losses.

Recommendation

Introduce a minimum amount out parameter for the `swap-x-for-y-fok`, `swap-y-for-x-fok`, `swap-x-for-y`, and `swap-y-for-x` swap functions in the `amm-pool-v3-helper` contract.

ClarityAlliance
Security Review

ALEX Lab

[L-07] AMM Operations Lack Deadline

Description

All AMM operations, such as swapping and adding or removing liquidity, currently lack a deadline parameter. Deadlines enable users to specify the time by which an operation must be executed; otherwise, it becomes invalid.

The absence of a deadline can lead to transactions being executed later than intended. This is particularly concerning for operations that mimic order book interactions, where Immediate or Cancel (IOC) actions must occur promptly. Miners might choose not to include the transaction in the next block due to resource usage or other considerations.

Recommendation

For all `amm-pool-v3-helper` entry point functions, incorporate a deadline parameter that is compared to the latest Stacks block time.

Currently, there is no mechanism to determine time-related information from code running in a transaction being executed in the latest block.

A workaround involves using a variable to denote the Stacks block time and considering it alongside the block timestamp when checking for staleness:

```
(define-constant STACKS_BLOCK_TIME u5)

(let ((block-timestamp (+ (unwrap-panic (get-stacks-block-info? time
  (- stacks-block-height u1)) STACKS_BLOCK_TIME))))
```

While the example snippet uses a constant 5 seconds to represent Stacks block time, a more robust implementation would involve making this value adjustable through governance. This would allow for adaptation to different scenarios and blockchain states.

In theory, Stacks blocks are minted approximately every 5 seconds. However, real-time data shows variations of up to tens of seconds between blocks. Additionally, there are instances where the Stacks blockchain temporarily halts block production and resumes after a significant delay.

A real-life example is the gap between Stacks blocks #242879 and #242880, which are 25 minutes apart. Any operation initiated in block `#242880` would use block `#242879` 's timestamp, resulting in a 25-minute discrepancy.

While this is not an ideal solution, it will help mitigate issues arising from delayed transaction executions.

CONTENTS

[L-08] Unsorted Tick Array Results in Suboptimal Swap Prices

Description

When a user performs a swap using the `swap-x-for-y`, `swap-y-for-x`, or `swap-routes` functions from the `amm-pool-v3-helper`, they must provide a list of ticks to be traversed in an attempt to fulfill their request.

The sequence of tick ranges within this array significantly impacts the quality of the swap. If the ticks are in ascending order and liquidity is available, the user will receive a more favorable price compared to when the ticks are in descending order.

For instance, consider a scenario where users swap tokens through ticks `[25, 30, 35]` with a price target of 6:

- Tick: 25, Price bounds: `[3.38635491, 3.55567265]`
- Tick: 30, Price bounds: `[4.32194233, 4.53803944]`
- Tick: 35, Price bounds: `[5.51601534, 5.79181610]`

If liquidity is available at tick 25, tokens will be swapped at the advantageous price range of `3.38 - 3.55`. If the swap is not fully completed, it will proceed to tick 30 at a less favorable price of `4.32 - 4.53`, and finally, if necessary, to tick 35 at an even less favorable price of `5.51 - 5.79`.

Conversely, if the ticks are provided as `[35, 30, 25]` and there is sufficient liquidity at tick 35 to complete the swap, the user would exchange all tokens at a higher price than if the ticks were in ascending order.

Recommendation

In the `swap-x-for-y` and `swap-y-for-x` functions of the `amm-pool-v3-helper`, either validate that the `ticks` are sorted and revert if they are not, or sort the ticks to ensure the best possible price for users.

CONTENTS

ClarityAlliance
Security Review

ALEX Lab

8.3. QA Findings

[QA-01] Token LP Contract Supports but Does Not Implement the SIP-13 Send Many Trait

Description

The V3 LP contract, `amm-liquidity-token-v3`, is an implementation of the SIP-13 semi-fungible token standard. It also supports the optional send-many specification.

Although the current implementation aligns with the send-many specification, the contract should also implement the trait itself using the `impl-trait` keyword. This would provide additional safety checks during deployment.

Recommendation

Implement the send-many trait in the `amm-liquidity-token-v3` contract.

CONTENTS

ClarityAlliance
Security Review

ALEX Lab

[QA-02] Reducing LP Position May Burn Dust Liquidity Without Affecting Balances

Description

The `amm-pool-v3` contract allows users to reduce tier LP positions by a percentage within a tick using the `reduce-position` function.

The implementation calculates the amount of LP tokens to burn and the corresponding amount of X and Y tokens to return to the user.

In extreme cases of withdrawing dust LP , it is possible to burn LP tokens without actually withdrawing any tokens, effectively acting as a donation.

These amounts are extremely small, on the order of a few nano units.

Recommendation

Ensure that both `balance-to-reduce-x` and `balance-to-reduce-x` are not simultaneously zero.

CONTENTS

[QA-03] Ambiguous Revert on Zero LP Amount Reduction

Description

The `amm-pool-v3` contract allows users to reduce tier LP positions by a specified percentage within a tick using the `reduce-position` function. This function ensures that the percentage is not zero and will revert if it is.

```
(asserts! (> percent u0) ERR-ZERO-PERCENT)
```

However, it does not check if the actual liquidity amount to be reduced (`balance-to-reduce`) is zero. The `balance-to-reduce` can be zero due to the division by 1e8 in the (`(mul-down lp-balance percent)`) operation, which occurs when attempting to remove negligible amounts of LP tokens at an almost zero percentage.

In such cases, the operation reverts with an `(err u1)` code, resulting from trying to burn a zero amount in the `amm-liquidity-token-v3::burn` function call. This revert complicates debugging failed transactions for users integrating at extreme cases.

Recommendation

Add an assertion in the `reduce-position` function to verify that `balance-to-reduce` is greater than zero. If it is not, the execution should revert with a custom error.

[QA-04] Implicit Minimum Token Transfer Amount

Description

The AMM pool is designed to operate with wrapper SIP-10 tokens, specifically deployed by the team. These wrapped tokens ensure a consistent decimal precision of 8, regardless of the underlying token.

All available pairs for swapping can be accessed via the AlexSDK::fetchSwappableCurrency API. At the time of writing this finding, the Alex SDK API supports 130 different tokens. The vast majority (with 3 exceptions) involve two variations of amount scaling before invoking the underlying `transfer` command:

```
transfer (/ (* amount (pow u10 (unwrap-panic (get-base-decimals))))
  (pow-decimals))

;; or

transfer (fixed-to-decimals amount)

;; with

(define-private (pow-decimals)
  (pow u10 (unwrap-panic (get-base-decimals))))

(define-private (fixed-to-decimals (amount uint))
  (/ (* amount (pow-decimals)) ONE_8))
```

Both variations are equivalent, with the amount being multiplied by 10 raised to the power of the underlying token's decimals, then divided by the scaled decimals (10e8). The 3 exceptions mentioned above have a 1:1 mapping and can be considered as having 8 decimals.

The division `(/ (* amount (pow-decimals)) ONE_8)` results in 0 if `amount * (pow-decimals) < ONE_8`. This zero amount will then trigger a code 3 error for a non-positive SIP-10 transfer amount.

For example, with the wSTX token wrapper, where 100 wSTX are equivalent to 1 STX, attempting to transfer any amount below 100 via `transfer-fixed` will revert.

Considering all 130 tokens, only 36% have 8 decimals, while the majority have fewer. There are even wrappers for tokens with 0 decimals.

Underlying Token Decimals	Minimum Transfer Amount	Count
0	100,000,000	5
1	10,000,000	1
2	1,000,000	4
3	100,000	12
4	10,000	1
5	1,000	1
6	100	57
7	10	3
8	1	46

The fewer the original token decimals, the higher the minimum transfer amount.

In all pool operations, when dealing with dust token amounts, or even up to 1 full unit (if tokens with 0 decimal count are used), users may encounter a confusing transfer 0 amount error message, complicating the debugging of failed transactions.

Recommendation

If desired, a universal method for determining the minimum can be implemented by modifying the custom SIP-10 trait to include the `get-base-decimals` function. Using this function, the minimum amount can be calculated before each transfer, reverting with a custom error if necessary.

However, the added overhead of determining the individual token minimum (where possible, as not all wrapped tokens have the `get-base-decimals` function) outweighs the benefits of such a change.

Therefore, we recommend acknowledging this finding and thoroughly documenting this behavior across the entire codebase, noting that: when working with dust amounts, callers may receive a `(err u3)` error if the minimum wrapper transfer amount is not reached.

CONTENTS

[QA-05] Absence of Events for Critical Actions

Description

In the `amm-liquidity-token-v3` contract, when a significant variable is modified, no event is emitted to notify off-chain monitoring systems.

The absence of events complicates protocol tracking for any third-party systems.

Recommendation

Incorporate a `print` command with relevant information for the following gated functions: `set-transferrable` , `set-token-base-uri` `set-name` , and `set-symbol` .

Note: Although `set-name` and `set-symbol` are generally not expected to change, it is advisable to emit events if they do.

[QA-06] Codebase Naming Improvement

Description

In the current codebase, there are instances where variable names do not fully represent the underlying functionality they are associated with or could be improved to be more descriptive.

1. `sunset`

The current implementation allows a pool to be “sunset,“ which deactivates the option to add liquidity. The term “sunset” is typically used to indicate the strategic decision to discontinue a particular product or service. The concept of “un-sunsetting” is not generally recognized.

However, the current implementation permits a pool to be “un-sunset,“ which may cause confusion due to the terminology.

2. `U64_MAX` and `U32_MAX`

The `U64_MAX` and `U32_MAX` constants, used in liquidity token ID operations, do not actually represent the maximum value for an unsigned integer stored in 64 bits (`U64_MAX`) or its 32-bit counterpart (`U32_MAX`).

```
(define-constant U64_MAX u18446744073709551616)
(define-constant U32_MAX 4294967296)
```

Both `U32_MAX` and `U64_MAX` are actually one more than the maximum value. `U64_MAX` would typically be `2^64 - 1` , but in this case, it is `2^64` , and `U32_MAX` would be `2^32 - 1` , but it is `2^32` .

3. `amm-pool-v3-helper`

The `amm-pool-v3-helper` contract serves as an entry point for several more complex pool operations. Generally, such contracts are referred to as `router` , not `helper` .

Recommendation

Implement the following changes:

- Rename `sunset` to `deactivated` throughout the `amm-pool-v3` pool contract.
- Rename `U32/U64_MAX` to a more accurate name, such as `U32/U64_CAP` or `U32/U64_SPACE` .
- Rename the `amm-pool-v3-helper` contract to `amm-pool-v3-router` .

CONTENTS

[QA-07] Overlapping or Discontinuous Error Code Ranges

Description

In the codebase, each contract should have a unique error code range to easily identify the originating contract of an error. However, the current implementation features overlapping and discontinuous ranges.

- The `ERR-NOT-AUTHORIZED (err u1000)` can be distinguished per contract.
- In `amm-liquidity-token-v3`:
    * The `ERR-TOO-MANY-POOLS (err u9001)` skips `9000` and other initial intervals, such as `4000`.
- In `amm-pool-v3`, there are multiple ranges: 1000-1999, 2000-2999, 3000-3999.

Overlapping ranges between contracts can cause confusion when debugging failed transactions. Having continuous ranges within a contract simplifies the process of extending and adding new errors.

Recommendation

Assign a distinct error range to each contract, starting from 10000 and incrementing for each error. The next contract should begin at 20000, the third at 30000, and so on. Ensure each contract maintains a single error range.

Note: This change should be applied to all contracts in the codebase, including those outside the audit scope, if possible.

CONTENTS

ClarityAlliance
Security Review

ALEX Lab

[QA-08] Add Preview-Operations Functions

Description

In general, for any AMM-type product, there are read-only helper functions that display the results of executing an operation on the AMM.

These “preview“ functions are used by users to assess whether a particular swap or liquidity operation is beneficial for them.

They also assist in selecting an appropriate slippage amount when using the `amm-pool-v3-helper::swap-routes` function or in determining the maximum or minimum price to use when executing swaps.

Currently, neither the AMM `amm-pool-v3` pool contract nor the router-like `amm-pool-v3-helper` contract provides such functions.

Recommendation

Consider adding preview functions for all AMM operations in the `amm-pool-v3` contract. These implementations would mirror the current functionality but without making any state changes.

[QA-09] Codebase Can Be Slightly Optimized

Description

There are instances within the codebase where a less execution fee-intensive implementation can be utilized.

Optimization #1: Use a `begin` block with an `assert` instead of the if/else unwrap logic

In the `amm-pool-v3::reduce-position` function, replace the if/else unwrap logic with a `begin` block using an `assert`.

```
-       (total-supply (unwrap! (if (> total-supply-unchecked u0)
-   (some total-supply-unchecked) none) ERR-POSITION-NOT-FOUND))
+       (total-supply (begin (asserts!
+   (> total-supply-unchecked u0) ERR-POSITION-NOT-FOUND) total-supply-unchecked))
```

Optimization #2: Save `(contract-of token-x/y-trait)` in `let` if used more than once

The `(contract-of token-x/y-trait)` call is executed twice in several functions: `add-to-position`, `reduce-position`, `swap-x-for-y-ioc`, and `swap-y-for-x-ioc`. In these cases, storing it as a `let` variable and reusing it would improve runtime costs, albeit with a slight increase in read length.

As some optimizations add adjacent costs by increasing the contract size while reducing the costs of a specific action, the following table shows actual gains per optimization on user-facing operations.

Optimization	`swaps`	`add-to-position`	`reduce-position`
#1	`"read_length": -8` `"runtime": -8`	`"read_length": -8` `"runtime": -8`	`"read_length": -8` `"runtime": -423`
#2	`"read_length": +13` `"runtime": -25389`	`"read_length": +13` `"runtime": -25389`	`"read_length": +13` `"runtime": -25389`

A positive value indicates that, after applying the change, the cost increased, while a negative value indicates it decreased.

From the table, it is clear that optimization #1 improves all operations, while optimization #2 slightly increases read overhead but provides a significant runtime optimization boost.

Recommendation

Implement the mentioned changes.

CONTENTS

ClarityAlliance
Security Review

ALEX Lab

[QA-10] Include Returned Token Amounts in reduce-position Output

Description

Currently, when the `reduce-position` function is executed, the output only includes the number of LP tokens burned. This amount is always equal to the specified percentage multiplied by the caller's held balance.

For external callers, it would be beneficial to also include the amounts of `X` and `Y` tokens received from the pool.

Recommendation

Modify the return type of `amm-pool-v3::reduce-position` to a tuple that includes the number of burned LP tokens, as well as the amounts of `X` and `Y` tokens removed from the pool.