SPYWOLF

Security Audit Report

Audit prepared for

SuiDex V3 CLMM

Completed on

May 9, 2026

TABLE OF CONTENTS

Executive Summary 01

Disclaimer 02

Engagement Details 03

Audit Scope 04

Methodology 05

Severity Classification 06

System Overview 07

Architectural Observations 08

Findings Summary 09

Detailed Findings 10

Verified Safe Components 11

Code Quality Observations 12

Recommendations Beyond Findings 13

EXECUTIVE SUMMARY

SpyWolf was engaged by the SuiDex team to conduct a comprehensive security audit of the SuiDex V3 concentrated liquidity market maker (CLMM) protocol. The audit covered both the on-chain Move smart contracts and the supporting off-chain API and indexer infrastructure.

The protocol is a CLMM-style automated market maker on the Sui blockchain, derived from established reference implementations (Uniswap V3, Cetus integer-mate, Momentum). It supports concentrated liquidity positions, multi-token reward emissions, flash loans, flash swaps, and a TWAP oracle. The on-chain package is deployed as immutable on Sui mainnet, with administrative functions held by a 2-of-4 multisig.

Findings Overview

- Critical: 0
- High: 0
- Medium: 1
- Low: 4
- Informational: 1

Audit Scope

- Smart contracts: suidex-v3-clmm at commit c1d9630 (~5,800 LOC across 32 modules)
- Backend services: suidex-v3-api at commit 1f254de (~1,400 LOC)
- Live mainnet package: 0xb5f5...e56 and associated shared objects
- Methodology: manual line-by-line review, differential analysis against reference implementations, and 12 simulated attack scenarios

Key Conclusions

The core CLMM mathematics, swap execution logic, fee accounting, and oracle implementation were verified to be correct and consistent with established reference implementations. Flash loan and flash swap re-entrancy protection via the hot-potato pattern is correctly implemented. No exploit paths were identified that would allow an attacker to drain user funds or manipulate the protocol's state outside of normal trading.

The single Medium-severity finding concerns the default minimum tick range parameter, which permits Just-In-Time (JIT) liquidity sandwich attacks against regular swappers. The four Low-severity findings are defense-in-depth recommendations covering silent-truncation behavior in fee and reward collection, a division-by-zero edge case in flash loan repayment, and minor robustness improvements.

Overall Risk Assessment: LOW

The codebase reflects a mature implementation of a well-understood AMM design with no critical defects. Recommended remediations are non-blocking and primarily aimed at strengthening robustness against future code changes.

DISCLAIMER

This security audit report (the "Report") has been prepared by SpyWolf (the "Auditor") for SuiDex (the "Client") in connection with the smart contract codebase identified in the Audit Scope section. This Report is provided for informational purposes and is not intended to be relied upon as financial, legal, or investment advice.

Scope Limitation. The Auditor's review was limited to the specific repositories, commit hashes, and files identified in §6 (Audit Scope). Any code, configuration, or infrastructure outside the defined scope, including but not limited to off-chain key management, multisig operational procedures, frontend applications, browser wallet integrations, third-party libraries beyond those explicitly reviewed, and any future modifications to the codebase, is excluded from the audit and the conclusions of this Report.

Point-in-Time Assessment. This Report reflects the state of the audited code at the specified commit hashes as of the audit period. Subsequent changes to the codebase, deployed packages, or surrounding infrastructure are not covered. The on-chain SuiDex V3 package is deployed as immutable on Sui mainnet; this Report does not extend to any successor deployments, additional pools created after the audit period, or other contracts integrating with the audited package.

No Guarantee of Security. A security audit, even one conducted with reasonable professional care, cannot guarantee that the audited code is free from defects, vulnerabilities, or unintended behavior. Smart contract security is an evolving discipline, and novel attack vectors may emerge after the date of this Report. The absence of findings in any area should not be interpreted as a guarantee of correctness.

Trust Assumptions. Per engagement terms, this audit operates under the explicit assumption that holders of administrative capabilities (AdminCap, VersionCap, pool admin, rewarder admin) are trusted parties acting in good faith and maintaining adequate operational security. Centralization risks and governance-related concerns were excluded from the audit scope at the Client's direction.

No Investment Advice. Nothing in this Report constitutes a recommendation to invest in, interact with, or otherwise transact with the audited protocol or any associated tokens. Readers should conduct their own due diligence.

Liability. To the maximum extent permitted by law, the Auditor disclaims all liability for any loss or damage arising from reliance on this Report. The Auditor's total liability is limited to the fees paid by the Client for this engagement.

ENGAGEMENT DETAILS (1)

Client: SuiDex
Auditor: SpyWolf
Engagement Type: Comprehensive Smart Contract Security Audit
Audit Period: April 28, 2026– May 8, 2026
Report Version: 1.0
Report Delivery Date: May 10, 2026

Project Information

Field	Value
Project Name	SuiDex V3 CLMM
Protocol Type	Concentrated Liquidity Market Maker (CLMM)
Blockchain	Sui Mainnet
Smart Contract Language	Move (Sui framework)
Backend Stack	TypeScript, Fastify, PostgreSQL
Deployment Status	Live on mainnet (immutable package)

Repositories Audited

Repository	Commit Hash
suidex-v3-clmm	c1d9630
suidex-v3-api	1f254de

Deployed Mainnet Package

0xb5f529c1dcda6580a61bf7ee9fbd524b50be62f11044d137c8202c8cbace9e56

ENGAGEMENT DETAILS (2)

Engagement Workflow

1. Initial scoping and code review
2. Architectural analysis and reference comparison
3. Manual line-by-line review of in-scope modules
4. Simulated attack scenarios and exploit testing
5. Off-chain component review (API and indexer)
6. Bytecode verification against deployed package
7. Findings documentation and report drafting
8. Client review and remediation discussion
9. Re-test of remediated findings (where applicable)
10. Final report delivery

Communication

All audit-related communications, findings disclosures, and remediation discussions were conducted directly between the Client and the Auditor through agreed private channels. This Report constitutes the final formal deliverable of the engagement.

Acknowledgments

The Auditor acknowledges the SuiDex team's responsiveness, technical clarity, and willingness to discuss design decisions in depth throughout the engagement. The provided audit brief, deployment artifacts, and supporting documentation materially aided the review process.

AUDIT SCOPE(1)

This audit covered the complete SuiDex V3 CLMM smart contract codebase together with the supporting off-chain API and indexer infrastructure. The scope was defined jointly with the Client at engagement kickoff and locked to specific commit hashes to ensure reproducibility.

In-Scope Repositories

Repository	Commit	Purpose	LOC
suidex-v3-clmm	c1d9630	On-chain CLMM smart contracts	~5,800
suidex-v3-api	1f254de	Off-chain API and event indexer	~1,400

On-Chain Modules Reviewed

Module	Description
actions/trade.move	Swap execution, flash loans, flash swaps
actions/liquidity.move	Position open/close, add/remove liquidity
actions/collect.move	Fee and reward collection
actions/create_pool.move	Pool creation logic
actions/admin.move	Administrative operations
storage/pool.move	Pool state, swap core, reserve management
storage/position.move	LP position state and updates
storage/tick.move	Tick state management
storage/tick_bitmap.move	Tick traversal bitmap
storage/global_config.move	Fee tier configuration
utils/tick_math.move	Tick ↔ sqrt price conversion

AUDIT SCOPE(2)

On-Chain Modules Reviewed (cont.)

Module	Description
utils/sqrt_price_math.move	Sqrt price arithmetic
utils/swap_math.move	Per-step swap computation
utils/liquidity_math.move	Liquidity delta math
utils/oracle.move	TWAP oracle
utils/bit_math.move	Bit manipulation helpers
utils/constants.move	Protocol constants
integer-mate/*	Signed integer math primitives
version/*	Version control
app.move, error.move	ACL, AdminCap, error codes

Off-Chain Components Reviewed

File	Description
src/api.ts	Fastify HTTP API endpoints
src/indexer.ts	Sui event polling and processing
src/v3-math.ts	Off-chain port of CLMM math
src/db/setup.ts	Database schema

AUDIT SCOPE(3)

Live Mainnet Objects Verified

Object	Address
Package (immutable)	0xb5f5...e56
GlobalConfig	0x4a04...5e83
Version	0x0999...4d4d
ACL	0xbe43...91c6
Live SUI/SUITRUMP Pool	0xdf8c...b284
Admin Multisig (2-of-4)	0x58ad...066f

Out of Scope

The following were explicitly excluded from this audit per engagement terms:

- Centralization and governance risk analysis
- Multisig key management and operational security
- Frontend applications and user interface code
- Browser wallet integrations
- Third-party libraries beyond those bundled in the audited repositories
- Future deployments, upgrades, or contracts not present at the audited commit
- Economic and tokenomic design of any specific token launched on the platform

METHODOLOGY(1)

The audit followed a structured, multi-phase methodology designed to provide thorough coverage of the codebase while validating findings through both static analysis and simulated execution scenarios.

Phase 1: Familiarization and Architectural Analysis

The auditor began by reviewing the Client-provided audit brief, deployment artifacts, and project documentation. The codebase architecture was mapped to identify module dependencies, trust boundaries, and external touchpoints. Reference implementations were identified for differential comparison: Uniswap V3 Core for the canonical CLMM design, Cetus integer-mate for the Sui-native math primitives, and Turbos and Momentum for Sui-specific architectural patterns.

Phase 2: Manual Line-by-Line Review

Each in-scope module was reviewed manually, line by line, with particular attention paid to modules flagged as Critical or High priority in the audit brief. The review focused on:

- Arithmetic correctness, including rounding direction, overflow handling, and fixed-point precision
- State transition consistency between pool, position, tick, and reserve accounting
- Access control and capability-based permission flows
- Resource management under Sui's Move object model
- Re-entrancy and atomicity guarantees via the hot-potato pattern
- Edge cases at boundary values (min/max ticks, zero liquidity, zero amounts, sqrt price limits)
- Event emission completeness for off-chain indexing

Phase 3: Differential Analysis

Critical math and logic modules were compared against established reference implementations to identify deviations. Where SuiDex V3 implements custom features beyond the reference design (multi-token reward emissions, flash loans, configurable minimum tick range factor, TWAP oracle integration), each custom feature received independent scrutiny without reliance on reference behavior.

Phase 4: Simulated Attack Scenarios

Twelve attack scenarios were constructed and analyzed to validate the protocol's behavior under adversarial conditions. Each scenario was evaluated by tracing the relevant code paths, the Move object lifecycle, and the resulting state transitions. Scenarios are documented in §11.

METHODOLOGY(2)

Phase 5: Off-Chain Component Review

The supporting API and indexer were reviewed for consistency with on-chain semantics, schema correctness, idempotency in event processing, and exposure of sensitive operations. The off-chain math library was checked for parity with the on-chain implementation.

Phase 6: Bytecode Verification

The audited source code was compiled with the Sui CLI and the resulting bytecode compared against the deployed mainnet package to confirm that the audited source corresponds to the on-chain deployment. Results are documented in Appendix B.

Phase 7: Findings Documentation and Re-Test

Each finding was documented with location references, impact analysis, a reproducible scenario, and a specific remediation recommendation. After the Client provided remediation responses, affected findings were re-tested and the report updated to reflect the final remediation status.

Tools Used

Manual review was the primary methodology. Supporting tools included the Sui CLI (sui move build, sui move test) for compilation and test execution, standard Unix text-processing utilities for cross-reference and pattern matching, and direct comparison against published reference implementations. No automated static analysis tools specific to Move are currently mature enough to be relied upon as a primary signal; their absence in this audit reflects the current state of the Move tooling ecosystem rather than a methodological gap.

SEVERITY CLASSIFICATION

Findings in this Report are classified by severity based on a combination of impact and likelihood. The classification system is designed to communicate both the technical seriousness of an issue and the realistic probability of exploitation, allowing the Client to prioritize remediation effort appropriately.

Severity	Description
Critical	Issues that allow direct theft or permanent loss of user funds, complete protocol compromise, or irreversible bricking of core functionality. Exploitation requires no special privileges and is straightforward to execute. Immediate remediation required.
High	Issues that allow theft or loss of funds under specific but realistic conditions, significant disruption to protocol operation, or substantial deviation from intended economic behavior. Exploitation may require specific market conditions or moderate technical sophistication. Remediation strongly recommended before further deployment.
Medium	Issues that cause measurable harm to users or the protocol under specific conditions but do not directly enable theft of principal. Includes economic inefficiencies that are systematically exploitable, defense-in-depth gaps, and design choices that materially weaken the security posture. Remediation recommended.
Low	Issues with limited direct impact, requiring unlikely conditions to manifest, or representing defense-in-depth concerns where current code is correct but lacks safeguards against future regressions. Remediation suggested for code robustness.

Determination Methodology

Severity is determined by evaluating two dimensions: **Impact** and **Likelihood**.

Impact considers the worst-case consequence if the issue is triggered or exploited:

- *Severe*: Direct loss of user funds, permanent protocol compromise, irreversible state corruption
- *Significant*: Partial loss of funds, sustained denial of service, material economic distortion
- *Moderate*: Limited fund loss under narrow conditions, recoverable disruption, accounting drift
- *Minor*: No direct fund loss, isolated edge case, robustness or quality concern

Likelihood considers how realistically the issue can be triggered in normal protocol operation:

- *High*: Triggerable by any user with no special access or market conditions
- *Medium*: Requires specific market conditions, moderate capital, or technical knowledge
- *Low*: Requires rare conditions, significant capital, coordinated action, or unusual integration
- *Theoretical*: Requires conditions that should not occur in practice but cannot be formally excluded

SYSTEM OVERVIEW(1)

SuiDex V3 is a concentrated liquidity market maker (CLMM) automated market maker deployed on the Sui blockchain. The protocol implements the design pattern established by Uniswap V3, in which liquidity providers (LPs) supply capital within custom price ranges rather than across the full price curve. This concentration improves capital efficiency for LPs and tightens spreads for traders, at the cost of increased complexity in pool accounting, position management, and tick-based state.

Core Components

The protocol consists of three layers: storage modules that hold persistent state, action modules that expose user-facing entry points, and utility modules that implement shared math and helper logic.

Layer	Modules	Responsibility
Storage	pool, position, tick, tick_bitmap, global_config	Pool state, LP position state, tick state, fast tick traversal bitmap, protocol-wide configuration
Actions	trade, liquidity, collect, create_pool, admin	User-facing entry points for swapping, liquidity management, fee/reward collection, pool creation, and administrative operations
Utilities	tick_math, sqrt_price_math, swap_math, liquidity_math, oracle, bit_math, constants, integer-mate/*	Tick/price conversion, swap step computation, liquidity-to-amount conversion, TWAP oracle, signed integer math
Control	app, version, error	AdminCap and ACL management, version gating, error codes

Design Choices

The implementation reflects several deliberate design choices that distinguish it from a vanilla Uniswap V3 port:

- **Q64 fixed-point representation** is used for square root prices and fee growth values, matching the Cetus and Turbos convention on Sui rather than the Q128 representation used by Uniswap V3 on Ethereum
- **Tick range** is bounded by ±443,636, matching Cetus rather than Uniswap V3's ±887,272 bound

SYSTEM OVERVIEW(2)

- **Multi-token reward emissions** allow each pool to distribute up to multiple reward tokens simultaneously, with per-position reward accrual tracked independently for each reward stream
- **Flash loans and flash swaps** are exposed via the hot-potato pattern, in which a non-droppable receipt object enforces atomic repayment within the same programmable transaction block (PTB)
- **TWAP oracle** is integrated into pool state, recording an observation array that downstream contracts can query for time-weighted average tick and seconds-per-liquidity values
- **Configurable minimum tick range factor** allows pool admins to enforce a minimum width on LP positions, restricting JIT liquidity strategies on a per-pool basis
- **Immutable package deployment** removes the upgrade attack surface entirely; the deployed bytecode cannot be modified after publication

On-Chain Architecture

The protocol is published as a single immutable Move package. Pool objects are shared, allowing any user to interact with them concurrently. Position objects are owned by their creator and represent transferable LP receipts that can be displayed as NFTs via Sui's Display framework. Administrative capabilities are represented as AdminCap and VersionCap objects, currently held by a 2-of-4 multisig.

Off-Chain Architecture

The supporting infrastructure consists of a Fastify-based HTTP API serving aggregated pool and position data, and a polling indexer that periodically fetches recent on-chain events from a Sui RPC endpoint and writes them to a PostgreSQL database. A separate TypeScript port of the on-chain math library supports off-chain calculations such as quote estimation and TVL aggregation.

Deployment Status

At the time of audit, the protocol is live on Sui mainnet with a single seeded pool (SUI/SUITRUMP) used by the team for stress testing under real economic conditions. Public pool creation has not been enabled. The Client has indicated that opening pool creation to general users is the intended next phase following remediation of audit findings.

ARCHITECTURAL OBSERVATIONS

This section documents structural observations about the codebase that, while not vulnerabilities, inform how the audit was conducted and how findings should be interpreted.

The codebase reflects an experienced implementation of an established design. Module boundaries are well-defined, with storage, actions, and utilities cleanly separated. Public entry points consistently apply version gating, pause checks, and pool verification before any state mutation, establishing a uniform defensive posture across user-facing functions. Internal package-only functions correctly use public(package) visibility, preventing external bypass of higher-level invariants.

Reference fidelity is strong. The core math primitives (tick math, sqrt price math, swap step computation, liquidity math) closely mirror their Uniswap V3 counterparts, and the integer-mate library is unmodified Cetus code. This adherence to known-good references substantially reduces the risk surface relative to a from-scratch implementation, and allows differential analysis to be a primary audit technique.

The hot-potato pattern is applied consistently for atomicity-critical flows. Flash loans, flash swaps, and any operation that lends out balances against future repayment use non-droppable receipt objects, forcing the entire borrow-and-repay sequence to complete within a single transaction. This pattern provides re-entrancy protection that is structurally stronger than the mutex-based approaches common in Solidity contracts, because the protection is enforced by the type system rather than by runtime checks.

The protocol introduces several extensions beyond a baseline CLMM: multi-token reward emissions with independent per-stream accounting, an integrated TWAP oracle, and a configurable minimum tick range factor that allows pool admins to restrict ultra-narrow LP positions. Each of these extensions introduces additional surface area that received independent review without reliance on reference behavior. The reward emission design is the most complex extension and was given particular attention; the math is conservative, with floored arithmetic used consistently to ensure that the sum of per-position claims cannot exceed funded reward balances.

The immutable package deployment is a deliberate and notable architectural choice. By forgoing upgrade capability entirely, the team removes a significant class of governance risk (malicious or buggy upgrades) at the cost of removing the ability to patch discovered issues post-deployment. This trade-off informs the severity classification of certain findings: defense-in-depth recommendations that would normally be Low severity remain Low, but their remediation cost is structurally higher because future fixes require redeployment rather than upgrade.

Off-chain infrastructure is functional but less mature than the on-chain code. The API and indexer are appropriate for a single-pool launch but contain assumptions (hardcoded token pricing, single-pool TVL logic) that will require attention before the protocol opens to general pool creation. These observations are reflected in the relevant findings.

Overall, the codebase is a competent implementation of a well-understood design with a clear separation between battle-tested core logic and team-specific extensions. The core is conservative; the extensions are where the audit's attention was concentrated.

FINDINGS SUMMARY

The following table summarizes all findings identified during this audit. Detailed descriptions, impact analysis, proof-of-concept scenarios, and remediation recommendations for each finding are provided in the subsequent subsections.

ID	Title	Severity	Status
M-01	Default min_tick_range_factor Permits JIT Liquidity Sandwich Attacks	Medium	Open
L-01	safe_withdraw Silently Truncates in Fee and Reward Collection	Low	Open
L-02	Reward Custodian Balance Invariant Not Enforced at Withdrawal	Low	Open
L-03	Flash Loan Repayment Aborts on Division-by-Zero When Pool Liquidity is Drained	Low	Open
L-04	Single-Pool Assumptions Hardcoded in API and Position Display	Low	Open
I-01	Zero-Amount Swap and Flash Loan Calls Produce No-Op Transactions	Informational	Open

Reading Guide

Findings are presented in descending order of severity. Each finding follows a standard structure: location references to the affected code, a description of the issue, impact analysis covering the conditions under which the issue manifests, a concrete scenario demonstrating the behavior, and a specific recommendation for remediation. Status reflects the remediation state at the time of final report delivery.

DETAILED FINDINGS

M-01: Default min_tick_range_factor Permits JIT Liquidity Sandwich Attacks

Severity: Medium
Category: Economic / MEV
Location: sources/storage/global_config.move, sources/storage/pool.move, sources/storage/tick.move

Description:
The min_tick_range_factor parameter sets the minimum width of an LP position as a multiple of the pool's tick spacing. The default value at pool creation is 1, which permits positions exactly one tick spacing wide.

This default matches Uniswap V3's reference behavior but leaves the protocol exposed to Just-In-Time (JIT) liquidity attacks, a well-documented MEV strategy in which an attacker briefly inserts ultra-narrow liquidity along a pending swap's price path to capture its fees, then removes the position immediately after.

Impact:
Because fee distribution within an active tick is proportional to liquidity share, an attacker who briefly dominates liquidity at the swap's tick captures the dominant share of fees.

- **Long-term LPs** see their fee share systematically diluted by JIT entrants on every large swap
- **Swappers** are not directly harmed on execution price, but the protocol's fee market becomes less attractive to genuine LPs, reducing passive liquidity over time
- **Protocol** experiences fee revenue flowing to searchers rather than committed capital, weakening the long-term LP incentive

Sui's transaction model differs from Ethereum's, but JIT remains achievable through colocated submission and observation of swap intents on integrating frontends. The economic logic is not blockchain-specific.

Scenario:

A pool has tick spacing 60, current tick 1000, and min_tick_range_factor = 1. An honest LP holds a wide position with liquidity 10,000.

An MEV searcher observes a pending swap moving the price from tick 1000 to 1020 and bundles:

1. Open position [960, 1020] with liquidity 990,000
2. Target swap executes
3. Remove the position, collecting accrued fees

During the swap, active liquidity is 1,000,000. The JIT position captures 99% of the fees; the honest LP receives 1% of what they would have earned. Repeated across swaps, this reallocates fees from passive LPs to searchers.

Recommendation:

The protocol exposes set_min_tick_range_factor for per-pool adjustment, allowing remediation without contract changes.

Immediate: Set min_tick_range_factor to at least 10 on all deployed pools and any pools created before broader remediation. This preserves capital efficiency for genuine LPs while substantially raising JIT capital cost.

Pre-launch of public pool creation: Raise the default applied at pool creation from 1 to a protocol-safe minimum. Consider scaling the default by fee tier — low-fee pools (where JIT is most profitable) receive higher defaults — and require explicit administrative action to lower the factor below a protocol-wide floor.

Several established Sui CLMMs ship with stricter defaults than the Uniswap V3 reference, reflecting industry consensus that the V3 default is too permissive for production deployments.

DETAILED FINDINGS

L-01: safe_withdraw Silently Truncates in Fee and Reward Collection

Severity: ☐ Low
Category: Defense-in-Depth / Accounting
Location: sources/storage/pool.move:888 (safe_withdraw), sources/storage/pool.move:629 (collect_fee), sources/storage/pool.move:642 (collect_reward)

Description:
The internal safe_withdraw helper caps any withdrawal at the available balance:

```
fun safe_withdraw<X>(balance: &mut Balance<X>, amount: u64): Balance<X> {
    let balance_val = balance::value<X>(balance);
    balance::split<X>(balance, math::min(amount, balance_val))
}
```

This helper is invoked from collect_fee and collect_reward. In both functions, the position is debited the full owed amount before safe_withdraw is called against the pool's reserves or reward custodian. If the available balance is lower than the owed amount, the position's accounting is decreased by the full amount but the user receives only the truncated amount. The shortfall is permanently lost with no event emitted.

Impact:
Under correct protocol operation, reserves and reward custodian balances should always cover outstanding claims. Fee growth accounting uses mul_div_floor consistently, and reward emission rates are computed conservatively, so claims accrue in the protocol's favor. As written, the truncation branch should not be reachable.
The risk is forward-looking. The function's silent-truncation behavior masks any future condition under which reserves fall below claims — whether introduced by a future code change, an unanticipated interaction between flash operations and fee accrual, or a subtle rounding regression. If such a condition ever materializes, affected users would lose funds with no on-chain signal that anything went wrong.
The package is immutable, so this behavior cannot be patched post-deployment. Defense-in-depth at the time of deployment is therefore disproportionately valuable.

Scenario:

Suppose a future state change or accounting drift leaves reserve_x at 1,000 while a position holds owed_coin_x of 1,500.

1. User calls collect_fee
2. position::decrease_owed_amount(position, 1500, 0) — position's owed_x goes to 0
3. take_from_reserves(pool, 1500, 0) calls safe_withdraw with amount=1500, balance_val=1000
4. User receives a Balance<X> of 1,000
5. The 500-unit shortfall is permanently lost; the user has no on-chain claim to recover it

The FeeCollectedEvent emitted reports the truncated amount, providing no signal that truncation occurred.

Recommendation:

Replace the min operation with an explicit assertion that fails loudly when the invariant is violated:

```
fun safe_withdraw<X>(balance: &mut Balance<X>, amount: u64): Balance<X> {
    assert!(balance::value<X>(balance) >= amount, error::insufficient_reserves());
    balance::split<X>(balance, amount)
}
```

This converts a silent loss-of-funds condition into a transaction abort, surfacing any underlying accounting drift immediately rather than allowing it to compound undetected. Aborting is preferable to silent loss because it preserves the user's claim — the position's owed amount remains debited only if the transaction succeeds.

If the team wishes to retain truncation behavior for any specific case, the alternative is to emit a dedicated ReserveShortfallEvent whenever truncation triggers, allowing off-chain monitoring to detect the condition. The assertion approach is preferred because it eliminates the risk entirely rather than relying on monitoring infrastructure.

DETAILED FINDINGS

L-02: Reward Custodian Balance Invariant Not Enforced at Withdrawal

Severity: ☐ Low
Category: Defense-in-Depth / Accounting
Location: sources/storage/pool.move:642 (collect_reward), sources/storage/pool.move:794 (update_reward_infos)

Description:
Each pool holds reward inventory in a per-token custodian balance, accessed via a dynamic field. Per-position rewards accrue through reward_growth_global and are claimed via collect_reward.

The emission math is conservative — the rate uses floored division, and per-position claims use mul_div_floor — so under correct operation the sum of claimable rewards cannot exceed the custodian balance. However, this invariant is not explicitly asserted at withdrawal. collect_reward debits the position's reward debt and then calls safe_withdraw, inheriting the silent-truncation behavior described in L-01.

Impact:
This is a forward-looking concern. The reward custodian is more sensitive than the main reserves because emission rates are recomputed on every update, pool liquidity transitions affect the accrual integral, and multiple reward streams accrue independently. If any combination of these conditions produced drift, a user would silently receive less than they accrued with no event signaling the shortfall.

Scenario:
A position holds coins_owed_reward = 5,000 while the custodian balance is 4,800.

1. User calls collect_reward
2. Reward debt reduced by full 5,000
3. safe_withdraw returns 4,800
4. User receives 4,800; the 200-unit shortfall is permanently lost
5. The emitted event reports only the truncated amount

Recommendation:
The assertion fix proposed in L-01 resolves this finding as well, since both rely on the same safe_withdraw helper.

Additionally, consider adding an explicit invariant check at the top of collect_reward:

```
assert!(
    balance::value(custodian) >= position::coins_owed_reward(position, reward_info_index),
    error::insufficient_reward_custodian()
);
```

This elevates the invariant to a first-class protocol property and produces a more diagnostic error than a generic balance-split abort, aiding off-chain monitoring.

DETAILED FINDINGS

L-03: Flash Loan Repayment Aborts on Division-by-Zero When Pool Liquidity is Drained

Severity: ☐ Low
Category: Robustness / Denial-of-Service
Location: sources/actions/trade.move:411 (flash_loan), sources/actions/trade.move:461 (repay_flash_loan)

Description:
The flash_loan function asserts pool::liquidity(pool) > 0 at the time of borrowing. The corresponding repay_flash_loan function does not re-check this condition. During repayment, the fee distribution computation divides the LP-share of the fee by the pool's current liquidity:

```
mul_div_floor(
    ((actual_fee_paid_x - protocol_fee_x) as u128),
    (constants::q64() as u128),
    liquidity,
)
```

If a caller borrows via flash_loan and then, within the same programmable transaction block, removes all of their own liquidity before calling repay_flash_loan, the pool's liquidity at repay time can be zero. The fee distribution computation then aborts with a division-by-zero error.

Impact:
This condition cannot be exploited for theft. The FlashLoanReceipt lacks the drop ability, requiring atomic repayment within the same transaction. If repay_flash_loan aborts, the entire transaction reverts and the borrowed balances are returned. The hot-potato safety property is preserved.

The impact is limited to denial-of-service against the caller's own transaction. A self-DOS is not a security concern in itself, but the abort surfaces as an opaque "division by zero" error rather than a descriptive protocol error, complicating debugging for integrators.

The condition is realistically reachable. Any LP who is also the only meaningful liquidity provider in a pool could trigger it accidentally by combining a flash loan with a position close in the same PTB. Integrators building automated strategies on top of the protocol may encounter this in testing.

Scenario:
A pool has total liquidity 100,000, all provided by a single LP. The LP constructs a PTB:

1. flash_loan for some amount of X and Y (succeeds; liquidity = 100,000 at borrow time)
2. remove_liquidity on their own position, reducing pool liquidity to 0
3. repay_flash_loan — aborts with division-by-zero in the fee distribution branch

The transaction reverts, the borrowed balance is returned, but the integration appears to fail mysteriously.

Recommendation:
Option A: Explicit assertion:

```
assert!(pool::liquidity(pool) > 0, error::insufficient_liquidity());
```

at the top of the fee-distribution branch in repay_flash_loan. This produces a clear, descriptive abort.
Option B: Route fees to protocol when liquidity is zero:
When liquidity == 0, skip the fee_growth_global update and route the entire fee (LP share included) to protocol_fee_x / protocol_fee_y. This allows the repayment to succeed and preserves the fee for future LPs or admin collection.

Option B is preferable because it avoids the DOS condition entirely while still honoring the fee economics. Option A is acceptable if the team prefers explicit failure over implicit re-routing.

DETAILED FINDINGS

L-04: Single-Pool Assumptions Hardcoded in API and Position Display

Severity: ☐ Low
Category: Forward-Compatibility / Multi-Pool Readiness
Location: suidex-v3-api/src/api.ts:132 (TVL calculation), suidex-v3-clmm/sources/utils/constants.move (position display image URL)

Description:
Two locations in the codebase hardcode assumptions specific to the current SUI/SUITRUMP launch pool. These assumptions are correct for the pool currently deployed but will produce incorrect or undesired behavior once the protocol is opened to general pool creation.

API TVL calculation. The /v3/pools endpoint computes pool TVL using a hardcoded ratio between the X and Y token prices:

```
result.set(row.pool_id, netX * suiPrice + netY * (suiPrice / 74000));
```

This expression assumes Y is SUITRUMP at a fixed price ratio of 1/74,000 relative to SUI. Applied to any other token pair, the computed TVL will be wrong, often by orders of magnitude.

Position display image URL. The position NFT display metadata embeds a hardcoded image URL pointing to https://suitrump.com/logocircle.png. This URL is bundled into the immutable package and cannot be changed post-deployment. All position NFTs across all future pools will display the SUITRUMP logo regardless of the underlying pool's token pair.

Impact:
Neither issue causes loss of funds. Both affect the protocol's readiness for the team's stated next phase — opening pool creation to general users.

The TVL miscalculation will produce misleading public statistics on any non-SUITRUMP pool, which can mislead users evaluating pool depth and fee opportunities. The display image issue is cosmetic but permanent: because the package is immutable, every position NFT minted on the protocol will reference the SUITRUMP logo regardless of pool composition. If the suitrump.com domain ever lapses or changes, all position NFTs lose their display image.

Scenario:
After public pool creation is enabled, a user creates a USDC/SUI pool and supplies liquidity. The /v3/pools endpoint reports a TVL value that misprices the USDC side as suiPrice / 74000, dramatically understating actual pool value. Frontends consuming this endpoint display incorrect information.

Separately, the user's position NFT — representing their LP share in the USDC/SUI pool — displays the SUITRUMP logo in their wallet.

Recommendation:
API TVL: Replace the hardcoded ratio with a per-token price lookup. Options include integrating a Sui price oracle (Pyth, Switchboard), querying a price aggregator, or maintaining an admin-configurable price map keyed by coin type. The API already fetches SUI price externally; extending this to other tokens is a modest change.
Position display image: Because the URL is bundled into the immutable package, the only on-chain remediation is to replace it in a future deployment. As an interim measure, the team can ensure the suitrump.com domain remains under their control indefinitely, or set up a redirect from the bundled URL to a more generic SuiDex-branded image. For any future redeployment, consider parameterizing the image URL per-pool, or hosting under a SuiDex-controlled domain that does not bind position branding to a specific token.

DETAILED FINDINGS

Informational

The following observations do not represent vulnerabilities and are documented for completeness. They identify minor robustness improvements, code quality opportunities, and defensive coding suggestions that the team may consider in future iterations of the codebase.

I-01: Zero-Amount Swap and Flash Loan Calls Produce No-Op Transactions

Location: sources/actions/trade.move

The flash_swap, flash_loan, and swap entry points do not assert that amount_specified > 0 (or that flash loan amounts are non-zero). Calls with zero amounts execute successfully but perform no meaningful work, returning zero-debt receipts and empty balances. The behavior is harmless but wastes gas and complicates integration debugging. Adding defensive assert! checks would produce clearer failure modes.

I-02: Documentation Discrepancy Between Audit Brief and Code Constants

Location: AUDIT_BRIEF.md, sources/utils/constants.move:73

The audit brief states the protocol fee share is 15% (150,000 in scaled units), while the code defines default_protocol_fee_share = 200000, equivalent to 20%. The code is authoritative; the brief should be updated to reflect the actual deployed value.

I-03: Opaque Error Surfaces from Generic Move Aborts

Location: Multiple

Several code paths can abort with generic Move errors (division-by-zero, balance-split underflow, vector-out-of-bounds) rather than protocol-defined error codes. While these aborts preserve safety, they produce poor diagnostic output for integrators. Wrapping the relevant operations in protocol-level assertions with descriptive error codes would meaningfully improve developer experience.

DETAILED FINDINGS

☐ Informational

I-04: SDK-Side Mitigations Documented as Protocol Mitigations

Location: AUDIT_BRIEF.md (H-1 mitigation)

The audit brief notes that SDK-side checks (e.g., "build-swap verifies liquidity > 0 before submission") mitigate certain on-chain abort conditions. SDK-side mitigations only cover users who interact through the SDK; direct PTB callers, smart contract integrators, and alternative frontends remain exposed to the underlying behavior. The brief should distinguish between protocol-level mitigations and client-level mitigations.

I-05: Hardcoded Cardinality and Magic Numbers

Location: sources/utils/constants.move, sources/utils/oracle.move

Several constants — observation cardinality bounds, fee tier scaling factors, tick range limits — are defined as inline magic numbers without documentation explaining their derivation or relationship to reference implementations. Adding comments tying these values to their source (Uniswap V3 reference, Cetus convention, protocol-specific design choice) would aid future maintainers and auditors.

I-06: Observation Cardinality Cannot Be Reduced

Location: sources/storage/pool.move:774 (increase_observation_cardinality_next)

The observation cardinality can only be increased, never decreased. This is intentional for oracle integrity but means that an admin who accidentally sets an excessive cardinality cannot reverse the change, permanently increasing per-pool storage cost. A note in operational documentation warning admins of the irreversibility would prevent accidental misconfiguration.

VERIFIED SAFE COMPONENTS(1)

This section documents components and behaviors that were specifically reviewed during the audit and confirmed to operate correctly. Inclusion here reflects active verification rather than absence of review. For a protocol deployed as immutable, explicit confirmation of soundness across the codebase's critical surfaces is part of the audit's value.

Core CLMM Mathematics

The tick math (utils/tick_math.move), sqrt price math (utils/sqrt_price_math.move), and per-step swap math (utils/swap_math.move) modules were verified against the Uniswap V3 reference implementation. The magic constants used in the bit-shift approximations of get_sqrt_price_at_tick and get_tick_at_sqrt_price match the reference values. Rounding directions in compute_swap_step correctly favor the protocol on input and the user on output, as expected.

Liquidity Math

The liquidity_math.move module's get_amount_for_liquidity, get_liquidity_for_amounts, and add_delta functions were verified against reference behavior. Rounding direction is selected correctly based on add-versus-remove operation, ensuring that LPs cannot extract more than they deposited via repeated add-and-remove cycles.

Integer Math Primitives

The integer-mate library (i32, i64, i128, u128, u256) is unmodified Cetus code, well-established and battle-tested across multiple production Sui deployments. Wrapping addition and subtraction (wrapping_add, wrapping_sub) are used appropriately in fee growth tracking, where wrap-around is the intended behavior.

Flash Loan and Flash Swap Atomicity

The hot-potato pattern is correctly applied. FlashLoanReceipt and FlashSwapReceipt lack the drop ability, requiring atomic resolution within the same programmable transaction block. Re-entrancy attacks via callback patterns common in EVM-style flash loans are not possible in this design. The receipts also bind the operation to a specific pool ID, preventing cross-pool repayment manipulation.

Cross-Position and Cross-Pool Guards

Every state-mutating operation that takes both a pool and a position invokes verify_pool to confirm the position belongs to the pool. Attempts to use a position with a non-matching pool abort cleanly. Type parameters <X, Y> further enforce token-pair binding at the type system level.

VERIFIED SAFE COMPONENTS(2)

Oracle Manipulation Resistance

The TWAP oracle's oracle::write function skips writes when the current observation's timestamp matches the new timestamp, ensuring that multiple swaps within the same transaction or block produce only one observation reflecting the start-of-transaction state. This eliminates intra-transaction oracle manipulation as an attack vector — an attacker cannot flash-swap to a manipulated price and then read that price from the oracle within the same atomic context.

Fee Growth Overflow Handling

Fee growth values use u128 with intentional wrapping arithmetic. Per-position deltas are computed via wrapping_sub against checkpoints, correctly recovering the true accumulated fee even when the global counter has wrapped. Overflow at the u128 scale is computationally infeasible under any realistic protocol usage.

Reward Over-Allocation Prevention

The reward emission rate is computed as (total_reward - total_reward_allocated) / duration with floored division, and per-position claims use mul_div_floor. The combination ensures that the integral of distributed rewards over time is always less than or equal to the funded amount. The invariant sum_of_claims <= total_reward holds across pause/unpause cycles, zero-liquidity transitions, and reward emission updates.

Position Update Ordering

In update_data_for_delta_l, fee and reward growth deltas are computed using the position's old liquidity before the new delta is applied. Tick state is updated after position state to preserve the correct accounting sequence. The clear-on-empty logic for ticks correctly differentiates between add and remove operations.

Slippage Protection

Both add_liquidity and remove_liquidity enforce user-supplied min_amount_x and min_amount_y parameters against the actual amounts produced by the operation. Users cannot have their funds taken at unexpected ratios due to mid-transaction price movement.

Tick Boundary Behavior

Operations at the minimum and maximum tick bounds (±443,636) were tested and behave correctly. Tick crossing at exact boundaries respects the direction of the swap, and the is_x_to_y direction logic in the swap loop correctly handles edge cases at tick boundaries.

VERIFIED SAFE COMPONENTS(3)

New Reward Stream Initialization

When a new reward token is added to a pool with existing positions and ticks, the missing-entry-defaults-to-zero pattern in tick reward growth tracking produces correct results. Because new rewards initialize with reward_growth_global = 0, the absence of a tick-level entry is consistent with a value of zero, and per-position reward calculations remain correct without requiring retroactive backfill.

Access Control Consistency

Administrative entry points consistently verify caller identity via the Acl object and app::get_* helpers. The AdminCap and VersionCap capabilities are passed by reference where appropriate, preventing capability theft via consumption. Package-only functions are correctly gated with public(package) visibility.

CODE QUALITY OBSERVATIONS(1)

This section documents observations about the codebase that fall outside the formal findings framework. These are not vulnerabilities and require no remediation, but they inform the overall picture of the codebase's maturity and may guide the team's prioritization in future development.

Strengths

The on-chain code demonstrates several markers of mature engineering:

- **Consistent structure across modules.** Storage, action, and utility responsibilities are cleanly separated. Public entry points apply version, pause, and pool-verification checks in a uniform order, making the defensive posture easy to verify by inspection.
- **Reference fidelity.** Where the protocol implements standard CLMM math, the code closely mirrors Uniswap V3 reference implementations, reducing the risk of subtle algorithmic mistakes.
- **Idiomatic Move usage.** The hot-potato pattern is applied where atomicity matters; capability objects are passed by reference where mutation is not required; package visibility is used appropriately. The code is recognizably written by someone familiar with the Move object model rather than a Solidity port.
- **Test coverage of core paths.** The included test suite covers the principal swap, liquidity, and version paths. Edge cases at tick boundaries and zero-liquidity transitions are exercised.

Areas for Improvement

Several observations point to opportunities for refinement rather than defects:

Off-chain code maturity. The API and indexer are functionally adequate for the current single-pool launch but are visibly less polished than the on-chain code. The schema-versus-runtime mismatch addressed by ALTER TABLE statements in indexer.ts suggests the schema definition has drifted from the live deployment. Consolidating the schema into a single source of truth (a migration system or a regenerated setup.ts reflecting the actual runtime state) would reduce future drift.

Inconsistent error specificity. Some operations abort with protocol-defined error codes via error::* helpers, while others abort with generic Move runtime errors (division by zero, balance split underflow). Standardizing on protocol-defined errors with descriptive codes throughout would improve diagnostic output for integrators and for future audits.

Magic numbers without inline derivation. Several constants are defined as numeric literals without comments tying them to their source. Examples include the tick range bound (443636), observation cardinality limits, and fee tier scaling factors. Adding brief comments referencing the derivation (Uniswap V3 reference, Cetus convention, protocol-specific design) would aid future maintainers.

RECOMMENDATIONS BEYOND FINDINGS (1)

Pre-Launch of Public Pool Creation

Before enabling general pool creation, the team should address the assumptions currently baked into the codebase that are valid only for the launch pool:

- Set min_tick_range_factor on all newly created pools to a value of at least 10 by default, ideally scaled by fee tier (see M-01)
- Replace the hardcoded TVL ratio in the API with a per-token price lookup mechanism (see L-04)
- Establish a permanent hosting solution for position NFT display images, ideally under a SuiDex-controlled domain rather than the current SUITRUMP-specific URL
- Document the operational expectation that admin keys remain held by the multisig, and publish the multisig address for community verification

Operational Monitoring

The protocol's immutable deployment elevates the importance of off-chain monitoring. The team should consider establishing dashboards and alerts covering:

- Reward custodian balances per pool and per reward token, with alerts if any custodian balance approaches the level of unclaimed rewards
- Reserve balance versus protocol fee accumulator consistency, providing early warning of any accounting drift
- Anomalous patterns in flash loan and flash swap usage that may indicate exploit attempts or integration bugs
- Indexer cursor lag relative to chain head, ensuring off-chain analytics remain current
- Position event throughput, with attention to JIT-pattern signatures (open and close in adjacent blocks within a tight tick range)

Test Suite Expansion

The existing Move test suite covers principal flows well. Expanding coverage in the following areas would strengthen confidence in edge case behavior and aid future audits or upgrades:

- Multi-reward-stream interactions, particularly when rewards are added or removed while positions hold accrued claims
- Pause and unpause transitions interleaved with active liquidity and swap operations
- Flash loan interactions with concurrent reward accrual and fee growth updates
- Tick boundary behavior under sequences of swaps that approach the minimum or maximum tick from both directions
- Position lifecycle scenarios involving full liquidity removal followed by re-addition, ensuring tick state is correctly cleared and re-initialized

RECOMMENDATIONS BEYOND FINDINGS (2)

Indexer and Schema Hardening

The off-chain indexer is currently maintained partly through runtime ALTER TABLE statements that patch schema drift at startup. The team should consolidate the schema into a single authoritative source, either by adopting a migration framework (such as node-pg-migrate or similar) or by regenerating setup.ts to match the current runtime schema and removing the runtime patches. Adding unique constraints on liquidity event tables would also resolve the de-duplication concern noted during audit.

Documentation for Integrators

As the protocol opens to broader use, integrators will need clear documentation of the on-chain interface. Specific items worth documenting include:

- The full lifecycle of FlashSwapReceipt and FlashLoanReceipt objects, including the conditions under which repay_* calls can fail
- The expected sequence of calls for collecting fees and rewards, including the role of update_data_for_delta_l with zero delta
- The economic implications of min_tick_range_factor for LPs and how administrators may adjust it
- The TWAP oracle's observation array semantics, including cardinality growth and the meaning of seconds_per_liquidity_cumulative

Future Audit Cadence

The protocol's immutable deployment limits the relevance of incremental upgrade audits, but several developments would warrant additional security review:

- Any successor deployment of the core package (whether for migration, fee adjustment, or feature expansion)
- Periphery contracts that build on the audited package, including routers, position managers, and integration adapters built by third parties
- Significant changes to the API or indexer that affect TVL reporting, position accounting, or external data exposure

Closing Remark

The protocol is well-positioned for its next phase. The recommendations above are oriented toward sustaining the quality already present in the on-chain code as the protocol expands to multi-pool operation and broader user adoption.

SPYWOLF

CRYPTO SECURITY

ABOUT US

We are a growing crypto security agency offering audits, KYCs and consulting services for some of the top names in the crypto industry.

- **OVER 1000 SUCCESSFUL CLIENTS**
- **MORE THAN 1000 SCAMS EXPOSED**
- **MILLIONS SAVED IN POTENTIAL FRAUD**
- **PARTNERSHIPS WITH TOP LAUNCHPADS, INFLUENCERS AND CRYPTO PROJECTS**
- **CONSTANTLY BUILDING TOOLS TO HELP INVESTORS DO BETTER RESEARCH**

Audits | KYCs | dApps
Contract Development

To hire us, reach out to
contact@spywolf.co or
t.me/joe_SpyWolf

FIND US ONLINE

SPYWOLF.CO

@SPYWOLFNETWORK

@SPYWOLFNETWORK

Disclaimer

This report shows findings based on our limited project analysis, following good industry practice from the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, overall social media and website presence and team transparency details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report.
While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER:
By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice.
No one shall have any right to rely on the report or its contents, and SpyWolf and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (SpyWolf) owe no duty of care towards you or any other person, nor does SpyWolf make any warranty or representation to any person on the accuracy or completeness of the report.
The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and SpyWolf hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, SpyWolf hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against SpyWolf, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts, website, social media and team.
No applications were reviewed for security. No product code has been reviewed.