SPYWOLF

Security Audit Report

Audit prepared for

SuiDex Locker

Completed on

Aug 22, 2025

TABLE OF CONTENTS

PROJECT INFORMATION

This audit is an add-on to a previous review, focusing specifically on the token_locker.move contract. This contract provides a mechanism for users to lock their VICTORY tokens for predefined durations to earn rewards from two distinct sources: ongoing VICTORY token emissions and a share of protocol revenue distributed in SUI.

The system's total reward pool is governed by the global_emission_controller.move contract, which acts as a "parent" utility contract, dictating the overall emission rates for the entire protocol. The token locker then subdivides its allocated rewards among stakers based on their chosen lock-up period and the amount staked.

Key Features

- **Variable Lock Periods:** Users can choose from four distinct lock-up durations: 7-day, 90-day, 1-year, and 3-year, each with different reward multipliers.
- **Dual-Reward System:**
    - **VICTORY Rewards:** Stakers earn VICTORY tokens from a dedicated reward pool. The distribution is managed by a MasterChef-style accumulator system (VictoryPoolAccumulator) that calculates rewards per share over time.
    - **SUI Revenue Sharing:** A portion of the protocol's revenue is distributed to lockers in the form of SUI. This system is managed through weekly epochs, where revenue is allocated and made claimable for eligible stakers.
- **Dynamic, Tiered Allocations:** The contract features a two-layer allocation system. First, it receives its total VICTORY reward budget from the global_emission_controller. Second, it subdivides this budget and the SUI revenue among the four lock periods based on admin-configurable percentages, incentivizing longer lock durations with higher reward rates.
- **Segregated Vaults:** The system uses three distinct vaults to ensure a clear separation of funds:
    - LockedTokenVault: Holds users' principal locked VICTORY tokens.
    - VictoryRewardVault: Holds VICTORY tokens deposited by the admin for reward distribution.
    - SUIRewardVault: Holds SUI tokens from protocol revenue for weekly distribution.
- **Admin-Controlled:** Privileged functions such as creating vaults, depositing rewards, and configuring allocation percentages are protected by an AdminCap, ensuring that only the authorized administrator can manage the contract's core parameters.

SCOPE OF AUDIT

The following document outlines the scope of the security audit conducted on the SuiDex token_locker.move smart contract. This audit is an add-on to a previous, broader review of the SuiDex ecosystem and specifically focuses on the token locker's internal logic, its integration with the protocol's tokenomics, and its overall security posture ahead of the mainnet launch.

The primary goal is to identify potential security vulnerabilities, logical inconsistencies, and areas where the implementation deviates from best practices, providing actionable recommendations for remediation.

The audit focused on the following components and areas:

- **Primary Contract Analysis:** A detailed, line-by-line review of the token_locker.move smart contract, including:
  - Logic for locking, unlocking, and managing user positions across all four lock periods (Week, 3-Month, Year, 3-Year).
  - The MasterChef-style accumulator system (VictoryPoolAccumulator) for calculating and distributing VICTORY token rewards.
  - The weekly epoch system for calculating and distributing SUI revenue-sharing rewards.
  - Access control mechanisms, particularly the use and transfer of the AdminCap.
  - State management of all user and pool data stored in tables.
  - Mathematical precision and potential for overflow/underflow errors in reward and fee calculations.
- **Key Integration Verification:**
  - Verification of the integration between the token locker and the global_emission_controller.move contract to ensure the correct and secure implementation of the documented tokenomics.
  - Interaction with the victorytoken.move contract for handling staking and reward tokens.
- **Analysis of Supporting Materials:**
  - A thorough review of the complete test suite for the token locker, including unit tests, integration scenarios, and Proof-of-Concept exploit verifications (token_locker_tests.move, poc.move, test_output.log).
  - Review of deployment scripts (complete_dex_setup.sh, create_wallet_pools.sh), build files (Makefile, Move.toml), and project documentation (TECH_DOCS.md, tokenomics.csv) to understand the intended production environment and economic model.
  - Cross-referencing findings with the provided SuiDex_Audit_Final.pdf report to confirm the status of previously identified vulnerabilities.

AUDIT METHODOLOGY

The audit was conducted using a multi-faceted approach to ensure comprehensive coverage:

1. **Static Analysis:** Manual line-by-line review of the token_locker.move source code and its direct dependencies.
2. **Test Suite Review:** Thorough examination of the provided test files (*_tests.move) and log outputs (test_output.log) to verify the correctness of the implementation and confirm fixes for known vulnerabilities.
3. **Comparative Analysis:** Cross-referencing the on-chain logic with project documentation (TECH_DOCS.md), tokenomics data (tokenomics.csv), and deployment scripts (*.sh) to identify any inconsistencies.
4. **Simulated Testing:** A logical walkthrough of contract functions and test scenarios to model the system's behavior under various conditions, including stress, edge cases, and known attack vectors.

FILES REVIEWED

File Name	Description
token_locker.move	(Primary Focus) The main contract for locking VICTORY tokens and managing the dual-reward (VICTORY & SUI) distribution system.
global_emission_controller.move	The "parent" contract that dictates the protocol-wide token emission schedule, rates, and decay, serving as the source of truth for tokenomics.
suifarm.move	The contract managing yield farming for both single-asset and LP token staking.
victorytoken.move	The definition of the VICTORY reward token, including minting and burning logic.
factory.move	Manages the creation and registry of liquidity pairs for the DEX.
pair.move	Implements the core AMM logic, liquidity management, and fee calculations for each individual pair.
router.move	Handles the routing of swaps, including multi-hop trades, and manages liquidity provision/removal logic.
library.move	A utility library containing mathematical formulas and helper functions used by the router and pair contracts.
fixed_point_math.move	A low-level library for high-precision u256 arithmetic, crucial for all financial calculations.

FOUND THREATS

🟥 High Risk ✅ FIXED

~~Victory rewards dilution on deposit (wrong update order)~~

Description:
When creating a Victory position, the code increases total_staked first and then updates the pool accumulator. This divides the past period's rewards by a larger total, underpaying existing stakers.

Impact:
Economic loss for existing stakers; rewards become time-of-deposit sensitive. Under certain deposit spikes, early stakers lose up to ~50% of the period's accrual for that update.

Code:

```
fun create_victory_position(
    locker: &mut TokenLocker, user: address, lock_id: u64, amount: u64, lock_period: u64,
    lock_end_time: u64, current_time: u64, global_config: &GlobalEmissionConfig,
    clock: &Clock, ctx: &mut TxContext
) {
    // ✅ STEP 1: Update total_staked FIRST        ← problematic order
    let accumulator_mut = get_victory_accumulator_mut(locker, lock_period);
    accumulator_mut.total_staked = accumulator_mut.total_staked + amount;

    // ✅ STEP 2: NOW update accumulator (with correct total_staked) ← too late
    update_victory_accumulator_for_period(locker, lock_period, global_config, clock, current_time);

    // ...calculate initial debt from the just-updated acc_per_share...
}
```

Recommendation:

Update the accumulator **before** mutating total_staked, then set the user's reward debt against the updated acc_per_share:

```
update_victory_accumulator_for_period(locker, lock_period, global_config, clock, current_time);
let acc_mut = get_victory_accumulator_mut(locker, lock_period);
acc_mut.total_staked = acc_mut.total_staked + amount;
// now compute initial debt using current acc_per_share
```

Implemented Fixes:

The order of operations in the create_victory_position function has been corrected. The code now correctly calls update_victory_accumulator_for_period *before* increasing the total_staked amount. This resolves the reward dilution vulnerability and ensures fair reward distribution for existing stakers.

FOUND THREATS

🟥 High Risk ✅ FIXED

~~Broken numeric literal (compile-blocking)~~

Description::
MAX_SAFE_ACCUMULATOR uses an invalid decimal literal with ... in the middle.

Impact:
The locker module will not build/publish (availability risk; also hides other issues until fixed).

Code:

```
const MAX_SAFE_ACCUMULATOR: u256 = 100000000000000000000000...000000000000000000000000000000000000000000000000u256; // 1e70
```

Recommendation:

Either remove the unused constant or replace with a valid single-line literal. Examples:

```
// If unused, delete it; otherwise:
const MAX_SAFE_ACCUMULATOR: u256 = 100000000000000000000000000000000000000000000000000000000000000000000000u256;
// or simply use u256::max_value if appropriate.
```

Implemented Fixes:

The MAX_SAFE_ACCUMULATOR constant, which was previously causing a compile-time error, has been completely **removed** from the token_locker.move contract. As the constant was unused, its removal is the cleanest and most effective solution.

FOUND THREATS

Medium Risk ✅ FIXED

~~Per-pool SUI claimed totals aren't hard-capped~~

Description:
Claim paths increment "claimed" counters but never cap them against the epoch's SUI pool totals. Floor rounding should keep sums ≤ pool totals, but a future change or new claim path could over-distribute.

Impact:
Potential overpayment from a pool if math or flows change; no invariant protects claimed ≤ pool_sui.

Code

```
fun update_epoch_pool_claimed(locker: &mut TokenLocker, epoch_id: u64, lock_period: u64, amount: u64) {
        let epoch = get_epoch_mut(locker, epoch_id);

        if (lock_period == WEEK_LOCK) {
            epoch.week_pool_claimed = epoch.week_pool_claimed + amount;
        } else if (lock_period == THREE_MONTH_LOCK) {
            epoch.three_month_pool_claimed = epoch.three_month_pool_claimed + amount;
        } else if (lock_period == YEAR_LOCK) {
            epoch.year_pool_claimed = epoch.year_pool_claimed + amount;
        } else {
            epoch.three_year_pool_claimed = epoch.three_year_pool_claimed + amount;
        };
    }
```

Recommendation:

Before incrementing, compute remaining = pool_sui - pool_claimed and add min(amount, remaining); assert that amount <= remaining to preserve the invariant.

Implemented Fixes:

Claim accounting now caps and asserts against per-pool remaining SUI before incrementing

```
let remaining = epoch.pool_allocations.week_pool_sui - epoch.week_pool_claimed;
assert!(amount <= remaining, E_INSUFFICIENT_POOL_BALANCE);
epoch.week_pool_claimed = epoch.week_pool_claimed + amount;
```

FOUND THREATS

Medium Risk ✅ FIXED

~~SUI epoch fairness depends on deposit-time totals~~

Description:
When weekly SUI is funded, the epoch snapshots pool totals at deposit time and uses that denominator for splits—while eligibility still requires full-week staking.

Impact:
If stake leaves mid-week (and thus isn't eligible), remaining users can still be diluted by the too-large denominator; they receive less SUI than economically fair for the ended week.

Code:

```
public entry fun add_weekly_sui_revenue(
        locker: &mut TokenLocker,
        vault: &mut SUIRewardVault,
        sui_tokens: Coin<SUI>,
        _admin: &AdminCap,
        clock: &Clock,
        ctx: &mut TxContext
    ) {
```

```
WeeklyPoolAllocations {
            epoch_id: current_epoch_id,
            week_pool_allocation: sui_week_allocation,
            three_month_pool_allocation: sui_three_month_allocation,
            year_pool_allocation: sui_year_allocation,
            three_year_pool_allocation: sui_three_year_allocation,
            week_pool_sui: week_sui,
            three_month_pool_sui: three_month_sui,
            year_pool_sui: year_sui,
            three_year_pool_sui: three_year_sui,
            // Snapshot current staking totals at time of allocation
            week_pool_total_staked: week_total_locked,
            three_month_pool_total_staked: three_month_total_locked,
            year_pool_total_staked: year_total_locked,
            three_year_pool_total_staked: three_year_total_locked,
        };
```

```
fun validate_full_week_staking_with_timestamps(user_lock: &Lock, week_start: u64, week_end: u64) {
        // User's lock must still be active after the week ends
        assert!(
            user_lock.lock_end >= week_end,
            ELOCK_EXPIRED_DURING_WEEK
        );

        // User must have staked BEFORE the week started (prevents gaming)
        assert!(
            user_lock.stake_timestamp < week_start,
            ESTAKED_DURING_WEEK_NOT_ELIGIBLE
        );
    }
```

Recommendation:

Either (a) only fund **after week end** and snapshot at end, or (b) compute shares using the set of **eligible** stake at week end, not deposit-time totals. Emit a finalization step if needed.

Implemented Fixes:

Funding now **requires the week to be finished** and **finalizes allocations** for that epoch:

```
assert!(current_time >= current_epoch.week_end_timestamp, E_WEEK_NOT_FINISHED);
assert!(!current_epoch.allocations_finalized, E_EPOCH_ALREADY_FINALIZED);
current_epoch.allocations_finalized = true;
```

FOUND THREATS

Medium Risk ✅ FIXED

~~Week-156 emission off-by-one vs tokenomics~~

Description:
The controller returns zero emissions for week 156 (week < TOTAL_EMISSION_WEEKS), while your tokenomics sheet shows week 156 still emitting. Locker rewards that depend on controller outputs inherit this.

Impact:
Final week Victory rewards are **0** per code/tests, diverging from the shared tokenomics schedule; this can cause user confusion and mis-accounting.

Code:

```
fun calculate_total_emission_for_week(week: u64): u256 {
        if (week <= 4) {
            // Bootstrap phase: 6.6 Victory/sec
            BOOTSTRAP_PHASE_EMISSION_RATE
        } else if (week == 5) {
            // Week 5: specific rate 5.47 Victory/sec
            POST_BOOTSTRAP_START_RATE
        } else if (week < TOTAL_EMISSION_WEEKS) {
            // Week 6+: apply 1% decay from week 5 rate
            let decay_weeks = week - 5;
            let mut current_emission = POST_BOOTSTRAP_START_RATE;

            // Apply 1% decay for each week after week 5
            let mut i = 0;
            while (i < decay_weeks) {
                current_emission = (current_emission * (WEEKLY_DECAY_RATE as u256)) / 10000;
                i = i + 1;
            };

            current_emission
        } else {
            // After week 156: no emissions
            0
        }
    }
```

```
public fun is_emissions_active(config: &GlobalEmissionConfig, clock: &Clock): bool {
        if (config.paused || config.emission_start_timestamp == 0) {
            return false
        };

        let current_week = calculate_current_week(config, clock);
        current_week < TOTAL_EMISSION_WEEKS && current_week > 0
    }
```

Recommendation:

Decide single source of truth. If emissions should include week 156, change week <= TOTAL_EMISSION_WEEKS and align is_emissions_active. Otherwise, update tokenomics/docs to match zero emission in week 156.

Implemented Fixes:

The controller now uses <= TOTAL_EMISSION_WEEKS consistently, with TOTAL_EMISSION_WEEKS: u64 = 156;

```
} else if (week <= TOTAL_EMISSION_WEEKS) {
...
assert!(target_week > 0 && target_week <= TOTAL_EMISSION_WEEKS, E_INVALID_WEEK);
```

FOUND THREATS

🟨 Low Risk ✅ Mitigated (design change)

~~Deferred SUI funding has no carry-over mechanism~~

Description:
If total staked is below the minimum when funding occurs, the SUI is parked in the vault and the function returns; there's no built-in mechanism to roll those funds into the next eligible epoch.

Impact:
Legitimate SUI funding can sit idle indefinitely, creating accounting/UX confusion and requiring manual intervention.

Code:

```
        // Store SUI for future epoch instead
        balance::join(&mut vault.sui_balance, coin::into_balance(sui_tokens));
        vault.total_deposited = vault.total_deposited + sui_amount;

        event::emit(FundingDeferred {
            reason: string::utf8(b"Insufficient total stake"),
            required: MIN_TOTAL_STAKE_FOR_FUNDING,
            current: total_staked,
            timestamp: current_time,
        });
        return
    };
```

Recommendation:

Add a scheduled "carry-over" or an explicit admin roll_deferred_sui_to_next_epoch that moves deferred balances into the first epoch that meets the minimum stake threshold (with an event).

Implemented Fixes:

Instead of "parking" SUI, the code **reverts** funding if the total staked is below threshold, so there's nothing to carry forward silently:

```
let total_staked = ...sum of pool totals...;
assert!(total_staked >= MIN_TOTAL_STAKE_FOR_FUNDING, E_INSUFFICIENT_TOTAL_STAKE);
```

FOUND THREATS

Low Risk

No rate-limit on SUI claim calls

Description:
claim_pool_sui_rewards has no minimum claim interval, unlike Victory claims which enforce MIN_CLAIM_INTERVAL.

Impact:
Users can spam SUI claim calls (even when already claimed), causing unnecessary gas/compute and noisy events, though not overpaying due to claim flags.

Code:

```
/// 🎯 PRODUCTION-READY: Claim SUI rewards with enhanced validation
public entry fun claim_pool_sui_rewards(
    locker: &mut TokenLocker,
    vault: &mut SUIRewardVault,
    epoch_id: u64,
    lock_id: u64,
    global_config: &GlobalEmissionConfig,
    clock: &Clock,
    ctx: &mut TxContext
) {
    let sender = tx_context::sender(ctx);
    let current_time = clock::timestamp_ms(clock) / 1000;
    validate_claim_allowed(global_config, clock);
```

Recommendation:

Optionally mirror Victory's throttling: track last SUI claim timestamp per (user, lock) and require current_time >= last + MIN_CLAIM_INTERVAL.

FOUND THREATS

Informational

Idempotent SUI claims via per-(epoch, lock) keys

The locker marks claims using a (epoch_id, lock_id) key, so calling a SUI claim function multiple times won't double-pay. This also makes indexing straightforward for off-chain analytics.

High-precision, overflow-safe Victory math

Victory reward accounting uses u256 with a 1e18 precision factor and safe mul/div helpers. Given the configured staking caps, intermediate products stay well below u256::MAX, so arithmetic overflow isn't a practical concern.

Rounding guarantees prevent over-distribution of SUI

Per-user SUI shares use floor rounding on proportional splits. By construction, the sum of all user payouts for an epoch is ≤ the pool's SUI for that epoch, leaving small, predictable "dust" rather than risking overpayment.

Full-week eligibility & claim gating

SUI rewards are only claimable if the lock existed before the week started and remains active through the week's end; claims are gated until after week end. This prevents mid-week "farm and dump" behavior.

Auto-settlement on unlock

The unlock flow settles any pending Victory/SUI rewards for the position before returning principal, reducing the chance of stranded rewards or user error during exit.

CONCLUSION

We re-reviewed the latest **Locker** code against your previous findings (05-A … 05-H) and validated the remediations on the hash-pinned artifacts you provided. All **Critical** and **High/Medium** issues have been addressed; the remaining items are **Low** or **Informational**.

Resolved

- **05-A** — Reward dilution on deposit: accumulator is updated **before** stake increases; user reward-debt is initialized from the updated acc_per_share.
- **05-B** — Broken numeric literal: corrected; module compiles cleanly.
- **05-C** — Per-pool SUI claimed totals: explicit per-pool **remaining** checks and caps added.
- **05-D** — Epoch fairness: SUI funding requires **week end** and **finalizes** allocations for that epoch.
- **05-E** — Emission window off-by-one: controller consistently uses <= TOTAL_EMISSION_WEEKS (156).

Mitigated (by design)

- **05-F** — Deferred SUI carry-over: funding now **reverts** when below the minimum total stake, eliminating silent deferral. (Optional enhancement: a formal carry-over mechanism if business requirements change.)

Low / Optional Enhancements

- **05-G** — SUI claim throttling: functionally safe; add a small per-(user, lock) claim interval if you want stricter rate-limits.
- (Optional) **Pagination** for bulk claim flows (e.g., (offset, limit)) to bound worst-case loop cost on very large histories.

Assurance & Readiness

Given the fixes above, we see **no outstanding Critical or High/Medium risks** in the Locker module. The contract set is **production-ready** assuming standard operational controls (admin hygiene, monitoring/alerting, and change management). Any subsequent code changes should be re-audited and re-pinned to preserve this assurance.

SPYWOLF

CRYPTO SECURITY

Audits | KYCs | dApps
Contract Development

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 700 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**

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.