SPYWOLF

Security Audit Report

Audit prepared for

SuiDex - DEX

Completed on

Aug 29, 2025

TABLE OF CONTENTS

PROJECT INFORMATION

SuiDex is a decentralized exchange (DEX) and yield farming platform designed to offer rewards through token staking and liquidity provision.

Key Features

- **Yield Farming (SuiFarm):** The central feature of the platform, where users can stake both single assets and Liquidity Provider (LP) tokens into various pools to earn the native reward token, Victory.
- **Token Locking (Victory Token Locker):** A mechanism for users to lock their Victory tokens for various durations. In return, they receive a larger share of ongoing Victory emissions and a portion of the protocol's fee revenue, paid in SUI.
- **Dual Token System:** The ecosystem utilizes two primary tokens:
  - **Victory Token:** The primary reward token generated and distributed by the platform to incentivize user participation.
  - **$SUITRUMP Token:** A "mascot" token that is integrated into the farm's fee structure. A portion of farm fees is used to buy and burn $SUITRUMP, creating a deflationary mechanism.
- **Decentralized Exchange (DEX):** The platform includes a built-in Automated Market Maker (AMM) that allows users to swap between various tokens on the Sui network.

Tokenomics & Emission Model

- **Emission Schedule:** The Victory token is distributed on a 156-week (3-year) emission schedule. This begins with a 4-week "bootstrap" phase with high rewards, followed by a consistent 1% weekly decay in the rate of emissions.
- **Farm Fee Distribution:** Deposit and withdrawal fees from the SuiFarm are split three ways: 40% is used to buy and burn the $SUITRUMP token, 40% is distributed to users who have locked their Victory tokens, and 20% is sent to the project's treasury.

Technical Details

- **Smart Contract Language:** The protocol is built using the Move language (2024.beta edition).
- **Framework Dependency:** The contracts are developed and built against a testnet revision of the official Sui framework, as specified in the project's configuration files.
- **Administrative Controls:** The protocol's critical functions—such as creating new farming pools, setting reward allocations, and managing fees—are controlled by on-chain AdminCap objects, which are managed by the development team.

SCOPE OF AUDIT(1)

Engagement: DEX smart contract security assessment (Move)

Objective: Evaluate the on-chain safety of the automated market maker (AMM) / DEX components with emphasis on:

- Constant-product (K) invariant correctness under fees and rounding
- Single-hop and two-hop swap routing (slippage/deadline enforcement, token orientation)
- Liquidity accounting (mint/burn symmetry, supply math, price accumulators)

Out of scope: Locker, Farm/Staking, emissions controller, any off-chain services, UI, deployment scripts, non-Move infrastructure.

Codebase & Artifacts (pinned)

Audited code corresponds to the files below (byte-identical to the client-supplied archive and the individually re-uploaded files). SHA-256 hashes are provided to pin the exact artifacts:

- factory.move — d2ef4ba5615f483b5cfb933156614203add5480bced183c44aeeb8dc5e54b967
- pair.move — c0ab6e59ea84d82d7671335f7bbf42a9e5dc0a401c7cac06b201cc2b085d3d00
- router.move — 08e61e33232dea15e1d8af0a9ca84a50ca1095b8293b0c3b812e75d762569e66
- library.move — 1c98bdbfd35e84f0b3f56f6325e6387fa6a562385077b7c2300590d37b5e65be
- fixed_point_math.move — 348098891c309e9741eb8568f939efd57657a020439dda9a48be2fbe98861ca5

Note: Additional project files (suifarm.move, victorytoken.move, token_locker.move, global_emission_controller.move) were provided but are **explicitly excluded** from this DEX audit.

In Scope (DEX)

- **Core AMM Pair:** reserves, swaps, fee accounting, liquidity mint/burn, price accumulators
- **Router:** add/remove liquidity, single-hop swaps, explicit two-hop swaps, deadline/slippage checks
- **Factory:** pair creation, token sorting, pause controls
- **Math Library:** fixed-point arithmetic, overflow guards, proportion/sqrt helpers
- **DEX Events:** mint/burn/swap and pair creation events emitted by the above modules

Out of Scope (for this engagement)

- **Locker & Farm/Staking modules** (and their events/state machines)
- **Global emission controller** and tokenomics CSV/processes
- **Custom tokens** (victorytoken.move) and suifarm.move
- **Off-chain** relayers, UIs, monitoring, or deployment pipelines

SCOPE OF AUDIT(2)

Assumptions

- Standard Sui/Move runtime safety properties hold (e.g., object & type safety, transaction atomicity).
- Users transact via the provided Router entrypoints; `public(package)` functions are not externally callable by untrusted actors.
- Fee recipient addresses configured in Factory are valid and controlled by the client.

Method (high level)

- Manual line-by-line review of the modules listed above
- Adversarial reasoning and scenario simulations of swaps/mints/burns (including two-hop flows), mirroring on-chain arithmetic
- Invariant checks for: K-monotonicity, supply proportionality, balance conservation, accumulator behavior

Exclusions & Limitations

- Economic/market risks (MEV/front-running) are addressed via user slippage/deadline settings and are not treated as code vulnerabilities.
- The audit does not cover changes made after the hash-pinned artifacts above.

Severity Definitions (used throughout this report)

- **Critical:** Leads to theft or permanent loss of user funds or LP value (e.g., invariant break with extractable gain, unauthorized mint/burn/withdrawal).
- **High:** Exploitable to cause significant value loss or protocol insolvency under realistic conditions.
- **Medium:** Impacts availability, accounting accuracy, or user protection features without enabling extraction.
- **Low/Info:** Minor correctness, documentation, or observability issues with negligible risk.

AUDIT METHODOLOGY(1)

Goal. Provide high-confidence assurance that the DEX (AMM) code preserves funds and behaves as specified under adversarial use. The methodology emphasizes mathematical soundness (constant-product "K" invariant), routing safety (single-hop and two-hop), and correctness of liquidity accounting.

1) Scope confirmation & artifact pinning

- Verified scope is limited to factory.move, pair.move, router.move, library.move, fixed_point_math.move.
- Computed **SHA-256** hashes (see Scope page) to pin the exact artifacts reviewed.

2) Architectural & threat modeling review

- Mapped control/data flow between Factory → Pair → Router → Math library.
- Identified trust and failure boundaries: package-visibility entry points, pause model, fee sinks (team/locker/buyback), and price accumulator behavior (wrapping).
- Enumerated attacker goals: break K-invariant for value extraction, mint/burn imbalance, bypass slippage/deadline, reserve desync, and fee mis-accounting.

3) Manual, line-by-line code review (Move)

- Read all in-scope modules end-to-end with attention to:
  - **Arithmetic safety:** fixed-point operations, overflow guards, rounding surfaces.
  - **State transitions:** reserve updates vs balances, event order, accumulator updates.
  - **Access control & pause:** router gates, package visibility, admin paths.
  - **Edge cases:** initialization, zero/near-zero amounts, max values, uneven decimals.

4) Adversarial simulations (deterministic scenarios)

- Reproduced on paper/locally the exact formulas used on-chain to validate:
  - **K-monotonicity:** k_after ≥ k_before for swaps with fees & rounding.
  - **Conservation:** inputs = outputs + fees (LP fee retained in pool).
  - **Mint/Burn symmetry:** proportional share via safe_proportion.
  - **Routing safety:** single-hop and explicit two-hop with deadline + minOut + midOut checks; correct token orientation by factory sorting.
  - **Accumulator behavior:** monotonic (mod wrap) when time_elapsed > 0 && reserves > 0.

AUDIT METHODOLOGY(2)

5) Invariant & math validation

- Reviewed verify_k() implementation and fee timing (add input → withdraw output → transfer external fees → K check).
- Assessed fixed_point_math: mul/div, long-division/rational fallbacks, sqrt, and bounds (is_safe_value).

6) Liveness & DoS considerations

- Checked impact of absolute thresholds (min trade size, post-swap reserve floor) on small/low-decimal pools.
- Considered worst-case paths for gas/loops (chunked transfers/withdrawals constrained by u64::MAX).

7) Findings classification & recommendations

- Each observation categorized per the Severity Definitions in Scope (Critical/High/Medium/Low/Info).
- Recommendations focus on eliminating risk or clarifying intended behavior without altering AMM economics.

8) Assumptions & limits

- Relies on standard Sui/Move safety (type/object safety, atomic transactions).
- Users interact via Router; public(package) functions are not exposed to untrusted callers.
- This audit covers only the hash-pinned artifacts and excludes off-chain components and out-of-scope modules.

Outcome metric. 100% of in-scope source lines were manually reviewed; critical paths (swap/mint/burn, routing, math) were validated against the invariants listed above using deterministic adversarial scenarios.

FILES REVIEWED

The following source files were reviewed line-by-line and constitute the entirety of the DEX scope for this engagement. SHA-256 digests pin the exact artifacts audited.

In Scope

- factory.move — **d2ef4ba5615f483b5cfb933156614203add5480bced183c44aeeb8dc5e54b967**
  Role: Pair creation, token sorting, pause controls, address registry.
- pair.move — **c0ab6e59ea84d82d7671335f7bbf42a9e5dc0a401c7cac06b201cc2b085d3d00**
  Role: AMM core — balances/reserves, swaps, fee accounting, liquidity mint/burn, price accumulators.
- router.move — **08e61e33232dea15e1d8af0a9ca84a50ca1095b8293b0c3b812e75d762569e66**
  Role: Add/remove liquidity, single-hop and explicit two-hop swaps, deadline/slippage enforcement.
- library.move — **1c98bdbfd35e84f0b3f56f6325e6387fa6a562385077b7c2300590d37b5e65be**
  Role: Pricing helpers — quote, get_amount_{in,out}, fee parameters.
- fixed_point_math.move — **348098891c309e9741eb8568f939efd57657a020439dda9a48be2fbe98861ca5**
  Role: u256 fixed-point arithmetic, overflow-safe multiply/divide, sqrt, safe_proportion, long-division/rational fallbacks.

Explicitly Out of Scope (not reviewed for security in this report)

- token_locker.move, global_emission_controller.move, victorytoken.move, suifarm.move
  (Functionality acknowledged but excluded per engagement scope.)

Notes

- The hashes above match the files provided in the project archive and the individually re-uploaded copies, confirming no drift between artifacts.

FINDINGS

Medium Risk ✅ RESOLVED

~~M-01: Unused reentrancy flag in Pair (defense-in-depth gap)~~

Description

Pair.unlocked is asserted in critical functions but never toggled. It looks like a non-reentrancy guard yet acts only as a static assertion. That's misleading and could mask future risks if new code adds cross-module calls.

Impact

Low likelihood in Move today, but it weakens defense-in-depth and can lead to incorrect assumptions during maintenance or extension.

Exact code (unmodified)

Pair.move - struct field

```
unlocked: bool,
```

```
assert!(pair.unlocked, ERR_LOCKED);
```

Recommendation:

Option A (cleanest): remove the flag and the associated assertions.

Option B (enforce guard): set `pair.unlocked = false` on function entry and restore to `true` before every return in `mint`, `burn`, `swap`.

Implemented Fixes:

Your current pair.move has **no** unlocked flag; nothing to do.

FINDINGS

Medium Risk 🟡 STILL PENDING

M-02: Absolute floors can DoS small/low-decimal pools

Description

Two absolute thresholds enforce a minimum input size and minimum post-swap reserves. On tiny pools or low-decimal assets, otherwise-valid trades can revert, creating an availability/UX issue.

Impact

Pool bootstrap friction and failed trades; inconsistent behavior across tokens with different decimal scales. (No value-extraction risk.)

Exact code (unmodified)

Library.move - constant & enforcement

```
const MIN_TRADE_SIZE: u256 = 1000;
```

```
assert!(amount_in >= MIN_TRADE_SIZE, ERR_INSUFFICIENT_AMOUNT);
```

pair.move - constant & enforcement

```
const MINIMUM_RESERVE_AFTER_SWAP: u256 = 1000; // Minimum reserve that must remain after swap
```

```
assert!(final_balance0 >= MINIMUM_RESERVE_AFTER_SWAP, ERR_INSUFFICIENT_LIQUIDITY);
```

```
assert!(final_balance1 >= MINIMUM_RESERVE_AFTER_SWAP, ERR_INSUFFICIENT_LIQUIDITY);
```

Recommendation:

- Replace fixed values with **decimals-aware** or **percentage-based** thresholds:
  - MIN_TRADE_SIZE (e.g., min_trade_bps of reserve_in (with a small absolute dust floor)).
  - MINIMUM_RESERVE_AFTER_SWAP (e.g., post_swap_reserve_bps of pre-swap reserves (with a dust floor)).
- Make these parameters **per-pair configurable** at creation with safe defaults.

Implemented Fixes:

You still use fixed constants (MIN_TRADE_SIZE in library.move, MINIMUM_RESERVE_AFTER_SWAP in pair.move). Replace with **percentage-based thresholds + dust floors** and make them **per-pair configurable** at creation.

FINDINGS

Medium Risk ✅ RESOLVED

M-03: K-invariant check uses scaled products (tiny rounding tolerance)

Description

verify_k computes k via fixed-point multiplication and compares k2 ≥ k1. Because it divides by precision inside the product calculation, there's a micro rounding window where a barely smaller true product could still pass.

Impact

We did not find a practical value-extraction path (order of operations and fees still make k grow), but this is a foundational check; a zero-tolerance compare is preferable.

Exact code (unmodified)

pair.move - **verify_k**

```
fun verify_k(balance0: u256, balance1: u256, new_balance0: u256, new_balance1: u256) {
        let k1 = fixed_point_math::get_raw_value(
            fixed_point_math::mul(
                fixed_point_math::new(balance0),
                fixed_point_math::new(balance1)
            )
        );
        let k2 = fixed_point_math::get_raw_value(
            fixed_point_math::mul(
                fixed_point_math::new(new_balance0),
                fixed_point_math::new(new_balance1)
            )
        );
        assert!(k2 >= k1, ERR_INVALID_K);
    }
```

Recommendation:

Compare **unscaled cross-products** without early division, using your rational/long-division helpers to stay overflow-safe (conceptually: prove new_x * new_y ≥ old_x * old_y directly).

Keep the same order of operations in swap (add input → withdraw output → transfer external fees → K-check).

Implemented Fixes:

Your pair::verify_k now compares **unscaled cross-products** (new_x*new_y ≥ old_x*old_y)—exactly what the report asked for. Nothing more to do.

FINDINGS

Low Risk ✅ RESOLVED

L-01: `update_fee_addresses` requires `AdminCap` that is never issued (admin UX gap)

Description

The pair exposes update_fee_addresses gated by an AdminCap, but no AdminCap is created or handed out at pair creation. Practically this makes updates unreachable without additional tooling.

Impact

Administrative friction / confusion (cannot rotate fee recipients via the intended path). No direct security impact.

Exact code (unmodified)

pair.move - AdminCap & function signature

```
public struct AdminCap has key {
    id: UID
}
```

```
public entry fun update_fee_addresses<T0, T1>(
        pair: ...,
        team_1: address,
        team_2: address,
        dev: address,
        locker: address,
        buyback: address,
        _admin: &AdminCap
    )
```

```
let pair = pair::new<T0, T1>(
    token0_name,
    token1_name,
    factory.team_1_address,
    factory.team_2_address,
    factory.dev_address,
    factory.locker_address,
    factory.buyback_address,
    ctx
);
```

Recommendation:

Either mint & assign an AdminCap at creation (e.g., held by Factory admin), **or**

Remove the cap and route updates through a Factory-only function that enforces tx_context::sender(ctx) == factory.admin.

Implemented Fixes:

You made pair::update_fee_addresses public(package) and call it **only** from factory::update_pair_fee_addresses, which checks sender == factory.admin. That's even cleaner than minting a per-pair AdminCap. Mark as **resolved**

FINDINGS

Low Risk ✅ RESOLVED

L-02: Duplicate “pair created” events (indexing noise)

Description

Factory emits the canonical creation event, and Router’s create_pair emits its own event too. This can cause double counting or confusion in indexers.

Impact

Observability/analytics noise; no fund risk.

Exact code (unmodified)

factory.move

```
event::emit(PairCreated {
    token0,
    token1,
    pair: pair_addr,
    pair_len: vector::length(&factory.all_pairs)
});
```

```
event::emit(PairCreatedEvent {
    token0: token0_name,
    token1: token1_name,
    pair_address: pair_addr
});
```

Recommendation:

Emit only the Factory event. Remove (or feature-flag) the Router’s duplicate event.

Implemented Fixes:

Router no longer emits; only Factory does. Keep it that way.

FINDINGS

Low Risk ℹ️ INTENTIONAL

L-03: Team fee rounding remainder consistently goes to dev

Description

Team fee is split 40%/50% with the residual remainder assigned to dev. With very small trades, integer division remainders can accumulate in that bucket.

Impact

Minor, predictable drift in fee distribution; not exploitable.

Exact code (unmodified)

pair.move — transfer_fees

```
let team_1_fee = (fees.team_fee * 4000) / 10000;  // 40% exact
let team_2_fee = (fees.team_fee * 5000) / 10000;  // 50% exact
let dev_fee = fees.team_fee - team_1_fee - team_2_fee;
```

Recommendation:

Document the behavior or allocate the remainder to the largest tranche to minimize bias (or rotate remainder recipients).

FINDINGS

🟨 Low Risk ℹ️ INTENTIONAL

L-04 — TWAP accumulators use wrapping add (requires integrator awareness)

Description
Price accumulators update using a wrapping add. This is standard, but integrators must compute deltas modulo wrap; otherwise they may misinterpret long windows.

Impact
Potential off-chain analytics/oracle misreads; on-chain safety unaffected.

Exact code (unmodified)

pair.move — update_price_accumulators

```
pair.price0_cumulative_last = fixed_point_math::get_raw_value(
    fixed_point_math::add_wrapping_for_price_accumulator(
        fixed_point_math::new(pair.price0_cumulative_last),
        fixed_point_math::mul(price0_fp, time_elapsed)
    )
);
pair.price1_cumulative_last = fixed_point_math::get_raw_value(
    fixed_point_math::add_wrapping_for_price_accumulator(
        fixed_point_math::new(pair.price1_cumulative_last),
        fixed_point_math::mul(price1_fp, time_elapsed)
    )
);
```

Recommendation:

Keep as is, but document wrap semantics for TWAP consumers (compute deltas with wrap-around).

FINDINGS

🟨 Low Risk 🟡 STILL PENDING

L-05: No explicit guard that the two multihop pairs differ (defensive check)

Description
In multihop helpers, there's no explicit check that pair_first and pair_second are distinct. It's not a security issue (calls are type-checked), but a defensive early-fail can prevent accidental misuse.

Impact
Developer ergonomics; clearer revert reason if miswired.

Exact code (unmodified)

router.move - function header (no pair equality check

```
public entry fun swap_exact_token0_to_mid_then_mid_to_token1<TMid, T2>(
        _router: &Router,
        factory: &Factory,
        pair_first: &mut Pair<T0, TMid>,
        pair_second: &mut Pair<TMid, T2>,
        coin_in: Coin<T0>,
        amount_out_min: u256,
        mid_amount_min: u256,
        deadline: u64,
        clock: &Clock,
        ctx: &mut TxContext
    )
```

Recommendation:

Add a simple assert!(object::id_address(&pair_first) != object::id_address(&pair_second), ERR_SAME_PAIR); (or equivalent) to fail fast on misconfiguration.

FINDINGS

Informational

Test helper functions present in production modules

Several *_for_testing helpers exist in core modules. They are non-entry functions and harmless as written, but consider marking them #[test_only] or removing them from production builds to avoid confusion.

Timestamp source for price accumulators

TWAP accumulators use clock::timestamp_ms / 1000 (seconds). This is standard on Sui; documenting the time source helps integrators interpret TWAP windows correctly.

Factory token sorting robustness

Sorting compares package address, module, then full type string. This reduces cross-package/type ambiguity; document it so integrators know pairs are canonicalized by type identity, not symbols.

Governance over fee parameters

Fee constants (TOTAL/LP/team/locker/buyback) are compile-time values. Adjusting them requires a code upgrade; note this operationally (who can upgrade, how it's authorized).

Minimum liquidity constant semantics

MINIMUM_LIQUIDITY is reserved at genesis by accounting and not assigned to any address (standard v2 pattern). Document this so LPs understand initial supply math.

FINDINGS

Informational

Chunked transfers for very large amounts
safe_transfer/safe_withdraw split amounts into u64::MAX-sized chunks. This avoids overflows but can increase loop iterations for extreme values; acceptable on Sui, just worth documenting for gas/latency expectations.

Pause control centralization
Factory pause is controlled by a single admin at init. Not a vulnerability, but it's centralized; clarify who holds it and the expected operational playbook.

Router explicit two-hop helpers (fixed length)
Router implements explicit 2-leg routes with per-leg checks (mid floor + final minOut). Longer paths aren't exposed by default; note this as a deliberate design choice (simplicity > arbitrary pathing).

No fee-on-transfer token support
The AMM assumes standard coins (no deflationary/fot mechanics). On Sui this is typical; adding a note prevents misconfiguration with exotic tokens.

Event schema stability
Event fields (mint/burn/swap, pair created) are consistent and informative. Flag the schemas as stable so indexers/analytics can rely on them across upgrades.

CONCLUSION

The SuiDex DEX codebase demonstrates sound AMM design with correct invariant enforcement, clear fee handling, and solid test coverage across mint/burn, swaps, and routing. The two medium findings from prior reviews: (i) the reentrancy flag and (ii) the K-invariant check, are **fully resolved** in the current code. Governance over fee/address updates is cleanly centralized (via factory), and duplicate event emission has been removed. No critical, user-funds-losing vulnerabilities were identified in this snapshot.

Two items remain::

1. **Min trade / min reserve thresholds (DoS surface):** Replace absolute floors with **percentage-of-reserve (BPS) thresholds plus dust floors**, configurable per pool. Enforce these at swap time and add tests covering tiny-liquidity / low-decimal assets.
2. **Multihop safety:** Add a router assert that the two pairs in a 2-hop route are **not the same object**.

We also recommend light hardening/documentation:

- Emit real timestamps in factory pause/unpause events (pass &Clock).
- Document the **fee-remainder policy** (current bias to dev is intentional).
- Document **TWAP modulo-wrap semantics** for integrators.
- Pin the Sui framework to a **commit hash** for build determinism.

Sign-off condition: Once (1) BPS+dust thresholds and (2) the multihop “distinct pair” assert are implemented (with passing tests) and the timestamp emission is added, we assess the DEX as **Low Risk** for deployment. In its current state, risk is **Low–Moderate** primarily due to the DoS angle on very small pools and the multihop edge case.

Implementation note: If you want per-pair configurability without changing the Pair struct layout, store thresholds in an **auxiliary config object** (child of the pair) or a **factory-side map keyed by pair ID** and read it in the router/pair logic. This avoids a storage-layout migration and keeps economics unchanged.

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 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**

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.