CAIRO SECURITY CLAN

STRKFARM

SECURITY ASSESMENT REPORT

MARCH 2025

Prepared for

STRKFARM

Contents

1 About Cairo Security Clan

Cairo Security Clan is a leading force in the realm of blockchain security, dedicated to fortifying the foundations of the digital age. As pioneers in the field, we specialize in conducting meticulous smart contract security audits, ensuring the integrity and reliability of decentralized applications built on blockchain technology.

At Cairo Security Clan, we boast a multidisciplinary team of seasoned professionals proficient in blockchain security, cryptography, and software engineering. With a firm commitment to excellence, our experts delve into every aspect of the Web3 ecosystem, from foundational layer protocols to application-layer development. Our comprehensive suite of services encompasses smart contract audits, formal verification, and real-time monitoring, offering unparalleled protection against potential vulnerabilities.

Our team comprises industry veterans and scholars with extensive academic backgrounds and practical experience. Armed with advanced methodologies and cutting-edge tools, we scrutinize and analyze complex smart contracts with precision and rigor. Our track record speaks volumes, with a plethora of published research papers and citations, demonstrating our unwavering dedication to advancing the field of blockchain security.

At Cairo Security Clan, we prioritize collaboration and transparency, fostering meaningful partnerships with our clients. We believe in a customer-oriented approach, engaging stakeholders at every stage of the auditing process. By maintaining open lines of communication and soliciting client feedback, we ensure that our solutions are tailored to meet the unique needs and objectives of each project.

Beyond our core services, Cairo Security Clan is committed to driving innovation and shaping the future of blockchain technology. As active contributors to the ecosystem, we participate in the development of emerging technologies such as Starknet, leveraging our expertise to build robust infrastructure and tools. Through strategic guidance and support, we empower our partners to navigate the complexities of the blockchain landscape with confidence and clarity.

In summary, Cairo Security Clan stands at the forefront of blockchain security, blending technical prowess with a client-centric ethos to deliver unparalleled protection and peace of mind in an ever-evolving digital landscape. Join us in safeguarding the future of decentralized finance and digital assets with confidence and conviction.

2 Disclaimer

Disclaimer Limitations of this Audit:

This report is based solely on the materials and documentation provided by you to Cairo Security Clan for the specific purpose of conducting the security review outlined in the Summary of Audit and Scoped Files. The findings presented here may not be exhaustive and may not identify all potential vulnerabilities. Cairo Security Clan provides this review and report on an "as-is" and "as-available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, occurs entirely at your own risk.

Inherent Risks of Blockchain Technology:

Blockchain technology remains in its developmental stage and is inherently susceptible to unknown risks and vulnerabilities. This review is specifically focused on the smart contract code and does not extend to the compiler layer, programming language elements beyond the reviewed code, or other potential security risks outside the code itself.

Report Purpose and Reliance:

This report should not be construed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. No third party should rely on this report for any purpose, including making investment or purchasing decisions.

Liability Disclaimer:

To the fullest extent permitted by law, Cairo Security Clan disclaims all liability associated with this report, its contents, and any related services and products arising from your use. This includes, but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Third-Party Products and Services:

Cairo Security Clan does not warrant, endorse, guarantee, or assume responsibility for any products or services advertised by third parties within this report, nor for any open-source or third-party software, code, libraries, materials, or information linked to, referenced by, or accessible through this report, its content, and related services and products. This includes any hyperlinked websites, websites or applications appearing on advertisements, and Cairo Security Clan will not be responsible for monitoring any transactions between you and third-party providers. It is recommended that you exercise due diligence and caution when considering any third-party products or services, just as you would with any purchase or service through any medium.

Disclaimer of Advice:

FOR THE AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, ACCESS, AND/OR USE, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

3 Executive Summary

This document presents the security review performed by Cairo Security Clan on the STRKFarm.

STRKFarm is a decentralized yield aggregator built on Starknet. It aims to maximize returns for users by automatically reallocating assets across various DeFi protocols. The platform leverages Starknet's scalability and low transaction costs to provide efficient yield farming opportunities. Users can deposit their assets into STRKFarm's vaults, which then optimize and manage the yield farming process. In simple terms, STRKFarm uses vaults to earn passive income for its users. Learn more from docs.

The audit was performed using

- manual analysis of the codebase,
- automated analysis tools,
- simulation of the smart contract,
- analysis of edge test cases

11 points of attention, where 0 is classified as Critical, 1 is classified as High, 0 is classified as Medium,2 are classified as Low,4 are classified as Informational and 4 are classified as Best Practices. The issues are summarized in Fig. 1.

This document is organized as follows. Section 1 About Cairo Security Clan. Section 2 Disclaimer. Section 3 Executive Summary. Section 4 Summary of Audit. Section 5 Risk Classification. Section 6 Issues by Severity Levels. Section 7 Test Evaluation.

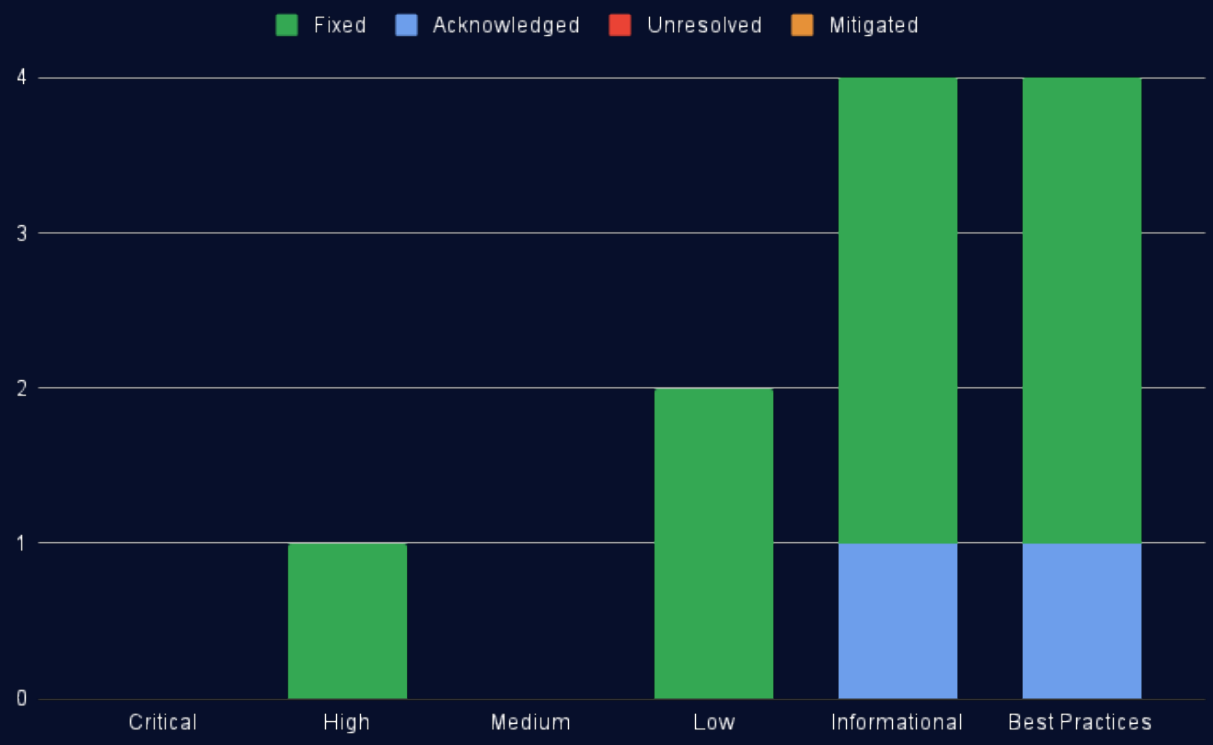

Fig 1: Distribution of issues: Critical (0), **High** (1), **Medium** (0), **Low** (2), **Informational** (4), **Best Practices** (4). **Distribution of status: Fixed** (9), **Acknowledged** (2), **Mitigated** (0), **Unresolved** (0).

4 Summary of Audit

Audit Type	Security Review
Cairo Version	2.8.4
Final Report	25/03/2025
Repository	strkfarm-contracts
Initial Commit Hash	45cdbcc7a9fc175846d4f2463cf845d73f56b23c
Documentation	Website documentation
Test Suite Assessment	Low

4.1 Completely Scoped Files

	Contracts
1	src/components/accessControl.cairo
2	src/components/common.cairo
3	src/components/ekuboSwap.cairo
4	src/components/erc4626.cairo
5	src/components/vesu.cairo
6	src/strategies/cl_vault/cl_vault.cairo
7	src/strategies/vesu_rebalance/vesu_rebalance.cairo

4.2 Partially Scoped Files

	Contracts
1	src/helpers/Math.cairo
2	src/helpers/safe_decimal_math.cairo
3	src/components/harvester/reward_shares.cairo
4	src/components/swap.cairo

In `src/helpers/Math.cairo`, the scope is restricted to the `max(...)` and `min(...)` functions.

In `src/helpers/safe_decimal_math.cairo`, the scope is restricted to the `address_to_felt252(...)`, `u256_to_address(...)`, `safe_subtract(...)`, `is_under_by_percent_bps(...)`, `fei_to_wei(...)` functions.

In `src/components/harvester/reward_shares.cairo`, the scope is restricted to the `get_additional_shares(...)` and `update_harvesting_rewards(...)` functions.

In `src/components/swap.cairo`, the scope is restricted to the `swap(...)` function.

4.3 Issues

	Findings	Severity	Update
1	Potential DoS in `harvest(...)` Due to Unwanted STRK Donations to `ConcLiquidityVault`	High	Fixed
2	`harvest(...)` can only theoretically called by anyone	Low	Fixed
3	Permissionless Call to `handle_unused(...)` Enables Potential Exchange Rate Manipulation	Low	Fixed
4	`safe_substract` function does not revert	Informational	Fixed
5	`harvest(...)` can revert if `rewardToken` is not STRK	Informational	Acknowledged
6	`VesuRebalance` Constructor requires deployer to have governor role	Informational	Fixed
7	Fee calculation may cause underflow.	Informational	Fixed
8	Redundant code	Best Practices	Fixed
9	Unused Storage Variable	Best Practices	Fixed
10	CEI Pattern Violation in `ConcLiquidityVault withdraw()`	Best Practices	Fixed
11	Unused `EkuboSwap` Component	Best Practices	Acknowledged

5 Risk Classification

The risk rating methodology used by Cairo Security Clan follows the principles established by the CVSS risk rating methodology. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely an attacker will uncover and exploit the finding. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to Motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Likelihood		
		High	**Medium**	**Low**
Impact	**High**	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Info/Best Practices

To address issues that do not fit a High/Medium/Low severity, Cairo Security Clan also uses three more finding severities: **Informational**, **Best Practices** and **Gas**

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to formally pass to the client;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Gas** findings are used when some piece of code uses more gas than it should be or have some functions that can be removed to save gas.

6 Issues by Severity Levels

6.1 High

6.1.1 Potential DoS in harvest(...) Due to Unwanted STRK Donations to `ConcLiquidityVault`

File(s): src/strategies/cl_vault/cl_vault.cairo

Description: The `ConcLiquidityVault` contract implements a strategy that manages users' liquidity positions in Ekubo. The vault collects trading fees and STRK token rewards, compounds them, and distributes them proportionally to liquidity providers through a reward-sharing mechanism.

The `harvest()` function is designed to claim STRK rewards, swap them for the underlying vault tokens (`token0` and `token1`), and reinvest these tokens back into Ekubo. This compounding mechanism is essential for users to maximize their yields. However, `harvest()` allows a malicious actor to disable the vault's reward harvesting functionality.

The function internally calls `update_harvesting_rewards()`, which contains a critical assertion:

```
assert(
    total_shares_u256 == 0 || shares.into() < total_shares_u256,
    'Invalid shares [3]'
);
```

This check ensures that the shares generated from a harvest don't exceed the total existing shares—a reasonable safety constraint under normal circumstances. However, if a malicious actor transfers or donates a substantial amount of STRK to the contract, the resulting shares calculation could exceed the total supply, causing this assertion to fail and blocking the harvest functionality entirely.

This happens because when shares are calculated during the harvest process, they are based on the new liquidity added but are checked against the current total shares supply before being added to it. Since the newly calculated shares aren't yet accounted for in the total shares supply at the time of this check, a large donation can cause the new shares to exceed the current total, triggering the assertion failure.

The malicious actor needs to transfer enough STRK such that, when processed through the harvesting mechanism, the calculated shares exceed or match the total supply of existing shares. For this, the actor will provide STRK worth more than or equal to the entire existing liquidity.

Even if the caller or admin attempts to call the function with the intention to reduce the STRK amount so that fewer shares are obtained, it will still revert due to the following strict check:

```
assert(
    swapInfo1.token_from_amount + swapInfo2.token_from_amount == STRK_bal,
    'invalid STRK balance'
);
```

This strict equality check prevents the caller from processing only a portion of the available STRK, forcing them to handle the full balance, including any malicious donations.

Attack Feasibility and Impact:

- The attack is particularly effective when the vault's total value locked (TVL) is low, such as during initial deployment or after significant withdrawals. - The amount of STRK needed to trigger this condition is lower when the vault has fewer existing shares. - The duration of this denial-of-service depends on vault activity and the price of STRK: - If the vault continues receiving deposits, total shares increase, potentially resolving the issue. - If the STRK price decreases, the attack becomes less effective as fewer shares are generated. - However, if STRK price rises, deposits slow down, or users withdraw funds, the DoS condition could persist indefinitely. - Users' earnings from STRK rewards are directly affected, as the vault cannot claim and reinvest STRK rewards, preventing them from benefiting from compounding.

Recommendation(s): Modify the `harvest()` function to only process legitimate STRK rewards from the distributor/claim contract, preventing arbitrary token deposits from disrupting the system.

Status: Fixed

Update from the client: Fixed in this commit.

6.2 Low

6.2.1 `harvest(...)` can only theoretically called by anyone

File(s): src/strategies/cl_vault/cl_vault.cairo, src/strategies/vesu_rebalance/vesu_rebalance.cairo

Description: Anyone can call the `harvest` function on both strategies as no access control mechanism has been implemented. However, to harvest correctly, cryptographic proofs must be passed as input. Since the protocol delegates harvest execution exclusively to the backend, making this function publicly accessible introduces potential attack vectors.

Recommendation(s): Implement an access control mechanism to restrict `harvest()` execution to authorized entities only.

Status: Fixed

Update from the client: Fixed in this commit.

6.2.2 Permissionless Call to `handle_unused(...)` Enables Potential Exchange Rate Manipulation

File(s): src/strategies/cl_vault/cl_vault.cairo

Description: The `handle_unused()` function in the `ConcLiquidityVault` contract can be called by anyone. When executed, this function deposits all `token0` and `token1` balances currently held by the contract into the Ekubo position, increasing the total liquidity tracked by the system. However, this liquidity increase occurs without minting any corresponding shares.

```
fn handle_unused(ref self: ContractState, swap_params: AvnuMultiRouteSwap) {
    // ...
    let token0_bal = ERC20Helper::balanceOf(pool_key.token0, this);
    let token1_bal = ERC20Helper::balanceOf(pool_key.token1, this);
    self._ekubo_deposit(this, token0_bal, token1_bal, this);
    // ...
}
```

This permissionless design introduces a potential attack vector, as it allows manipulation of the vault's share-to-liquidity ratio. An attacker could transfer tokens directly to the contract and call `handle_unused()` to alter the ratio. While this manipulation may not provide direct financial gains (as the attacker would need to provide the tokens themselves), it introduces unpredictability that could be exploited in more complex manipulation strategies, potentially harming users and the protocol.

Recommendation(s): Implement access control for the `handle_unused()` function to prevent unauthorized liquidity injections.

Status: Fixed

Update from the client: Fixed in this commit.

6.3 Informational

6.3.1 `safe_substract` function does not revert

File(s): src/helpers/safe_decimal_math.cairo

Description: The `safe_subtract()` function should revert if the first value is lower than the second. However, instead of reverting, it incorrectly returns zero.

```
pub fn safe_subtract(a: u256, b: u256) -> u256 {
    if a < b {
        return 0;
    }
    a - b
}
```

Recommendation(s): Consider reverting when `a < b`. If a revert is not desired, consider returning a `Result` type to handle errors properly.

Status: Fixed

Update from the client: This was intentionally written this way since a u256 overflow already triggers an error. Renamed the function to `non_negative_sub` for better clarity and to reflect its purpose more accurately. Update commit.

6.3.2 `harvest(...)` can revert if `rewardToken` is not STRK

File(s): src/strategies/cl_vault/cl_vault.cairo

Description: The `harvest()` function in the `ConcLiquidityVault` contract contains a logical inconsistency that causes it to revert when the claimed reward token is anything other than STRK. The function is designed to claim rewards from an Ekubo claim/distributor contract, convert those rewards to vault tokens (`token0` and `token1`), and add liquidity back to the position. The issue arises due to the conflicting usage of the `swapInfo1` parameter in two different contexts.

When a reward token other than STRK is claimed, the `simple_harvest()` function internally calls `check_and_swap_harvest()`, which expects `swapInfo1` to contain information for swapping from the non-STRK reward token to STRK. The validation in `check_and_swap_harvest()` requires:

```
assert(swapInfo.token_from_address == rewardToken, 'Invalid token from address');
assert(swapInfo.token_to_address == baseTokenAddress, 'Invalid token to address');
```

However, immediately after `simple_harvest()` completes in the `harvest()` function, there are contradictory validation checks that assume `swapInfo1` is configured to swap STRK to `token0`:

```
assert(
    swapInfo1.token_from_address == constants::STRK_ADDRESS(),
    'invalid token from address [1]'
);
assert(swapInfo1.token_to_address == token0, 'invalid token to address [1]');
```

These conflicting requirements create an impossible situation when the reward token is not STRK. The same `swapInfo1` cannot simultaneously be configured for swapping a non-STRK token to STRK (in `simple_harvest()`) and for swapping STRK to `token0` (in `harvest()`).

As a result, the contract is unable to handle non-STRK rewards, leading to transaction failures due to these mutually exclusive validation requirements.

Recommendation(s): Consider implementing logic to properly handle non-STRK rewards.

Status: Acknowledged

Update from the client: For now, its intentionally designed to support STRK only. Added a condition to check there is non-zero STRK rewards to ensure the same. Update at commit.

6.3.3 `VesuRebalance` Constructor Requires Deployer to Have Governor Role

File(s): src/strategies/vesu_rebalance/vesu_rebalance.cairo

Description: The `VesuRebalance` contract constructor calls external functions `set_allowed_pools()` and `set_settings()`, both of which contain `assert_governor_role()` checks. This creates a deployment restriction since the constructor can only be invoked by an address that already has the GOVERNOR role in the specified `AccessControl` contract.

```
#[constructor]
fn constructor(
    ref self: ContractState,
    asset: ContractAddress,
    access_control: ContractAddress,
    allowed_pools: Array<PoolProps>,
    settings: Settings,
    vesu_settings: vesuStruct,
) {
    // ...
    self.set_allowed_pools(allowed_pools);
    self.set_settings(settings);
    // ...
}
```

Recommendation(s): Consider using an internal function to avoid this deployment restriction.

Status: Fixed

Update from the client: Fixed in this commit.

6.3.4 Fee Calculation May Cause Underflow

File(s): src/strategies/vesu_rebalance/vesu_rebalance.cairo

Description: Fee calculation on `_collect_fees(...)` function may cause underflow because `DEFAULT_INDEX` value is used which has to be used for 18 decimal values.

Recommendation(s): Consider multiplying `total_supply` with index for proper fee calculation.

Status: Fixed

Update from the client: This issue detected by client, Fixed in PR #7 and PR #9

6.4 Best Practices

6.4.1 Redundant code

File(s): src/strategies/cl_vault/cl_vault.cairo, src/components/ekuboSwap.cairo

Description: The `ConcLiquidityVault` contract and the `ekuboSwap` component exhibit code redundancy in the following instances:

1. The code statement `let n_routes = routes.len();` in the function `get_nodes()` is used twice.

```
pub fn get_nodes(routes: Array<Route>, core: ICoreDispatcher) -> Array<RouteNode> {
    // ...
    let n_routes = routes.len();
    assert(n_routes > 0, 'EkuboSwap: no routes');
    let mut nodes: Array<RouteNode> = array![];
    let n_routes = routes.len();
    // ...
 }

```

2. In the `deposit()` function of the `ConcLiquidityVault` contract, redundant code can be simplified. Currently, the function calculates shares through a two-step process:

```
let liquidity = self._max_liquidity(amount0, amount1);
let shares = self._convert_to_shares(liquidity.into());

```

This is inefficient because the contract already has a public `convert_to_shares()` function that performs the same calculation:

```
fn convert_to_shares(self: @ContractState, amount0: u256, amount1: u256) -> u256 {
    let liquidity = self._max_liquidity(amount0, amount1);
    return self._convert_to_shares(liquidity.into());
}

```

Recommendation(s): Consider removing redundant code in both functions by eliminating the duplicated `n_routes` calculation in `get_nodes()` and replacing the two-step share calculation in `deposit()` with a direct call to the existing `convert_to_shares()` function.

Status: Fixed

Update from the client: Fixed in this commit. Haven't modified the two step calculation in deposit, because the output variable (liquidity) of step one is used below for an assert.

6.4.2 Unused Storage Variable

File(s): src/strategies/cl_vault/cl_vault.cairo

Description: The state variable `ownable` from OpenZeppelin's `OwnableComponent` remains unused. This component was likely introduced for ownership management but is redundant, as the contract already implements a comprehensive role-based access control system through the `AccessControl` contract via `CommonComp`.

```
#[storage]
struct Storage {
  //...
  #[substorage(v0)]
  ownable: OwnableComponent::Storage,
  //...
}
```

Recommendation(s): Consider removing this storage variable along with the associated imports.

Status: Fixed

Update from the client: Fixed in this commit.

6.4.3 CEI Pattern Violation in `ConcLiquidityVault withdraw()`

File(s): src/strategies/cl_vault/cl_vault.cairo

Description: The `withdraw()` function in the `ConcLiquidityVault` contract violates the Checks-Effects-Interactions (CEI) pattern by performing token transfers before updating the contract's state (burning shares).

```
fn withdraw(
    ref self: ContractState, shares: u256, receiver: ContractAddress
) -> MyPosition {
    //...
    ERC20Helper::transfer(pool_key.token0, receiver, amt0.into());
    ERC20Helper::transfer(pool_key.token1, receiver, amt1.into());

    self.erc20.burn(caller, shares);
    //...
}
```

Recommendation(s): Consider reordering the operations in the `withdraw()` function to follow the CEI pattern.

Status: Fixed

Update from the client: Fixed in this commit.

6.4.4 Unused `EkuboSwap` Component

File(s): src/components/ekuboSwap.cairo

Description: The codebase contains a fully implemented `EkuboSwap` component with functionality for performing token swaps through the Ekubo protocol. However this component is not utilized in either the `ConcLiquidityVault` or `VesuRebalance` contracts, which instead rely on Avnu swaps.

Recommendation(s): Consider either using the `EkuboSwap` component or removing it to simplify the codebase.

Status: Acknowledged

Update from the client: We retain `ekuboSwap` component for now for any future use.

7 Test Evaluation

7.1 Compilation Output

```
scarb build
   Compiling snforge_scarb_plugin v0.32.0 (github.com/foundry-rs/starknet-foundry?tag=v0.32.0#3817c903b640201c72e743b9bbe70a97149828a2)
    Finished `release` profile [optimized] target(s) in 1.01s
   Compiling lib(strkfarm_contracts) strkfarm_contracts v1.0.0 (/strkfarm-contracts/Scarb.toml)
warn: Usage of deprecated feature `"deprecated-list-trait"` with no `#[feature("deprecated-list-trait")]` attribute. Note: "Use `starknet::storage::Vec`."
 --> /strkfarm-contracts/src/strategies/vesu_rebalance/vesu_rebalance.cairo:28:42
    use alexandria_storage::list::{List, ListTrait};
                                   ^*******^

   Compiling starknet-contract(strkfarm_contracts) strkfarm_contracts v1.0.0 (/strkfarm-contracts/Scarb.toml)
warn: Usage of deprecated feature `"deprecated-list-trait"` with no `#[feature("deprecated-list-trait")]` attribute. Note: "Use `starknet::storage::Vec`."
 --> /strkfarm-contracts/src/strategies/vesu_rebalance/vesu_rebalance.cairo:28:42
    use alexandria_storage::list::{List, ListTrait};
                                   ^*******^

    Finished `dev` profile target(s) in 2 minutes
```

7.2 Tests Output

```
     Running test strkfarm_contracts (snforge test --max-n-steps 30000000)
warn: failed to open local registry cache, trying to recreate it

Caused by:
    Database already open. Cannot acquire lock.
   Compiling snforge_scarb_plugin v0.32.0 (github.com/foundry-rs/starknet-foundry?tag=v0.32.0#3817c903b640201c72e743b9bbe70a97149828a2)
    Finished `release` profile [optimized] target(s) in 0.24s
   Compiling test(strkfarm_contracts_unittest) strkfarm_contracts v0.1.0 (strkfarm-contracts/contracts/Scarb.toml)
warn: Usage of deprecated feature `"deprecated-list-trait"` with no `#[feature("deprecated-list-trait")]` attribute. Note: "Use `starknet::storage::Vec`."
 --> /strkfarm-contracts/contracts/src/strategies/vesu_rebalance/vesu_rebalance.cairo:28:42
    use alexandria_storage::list::{List, ListTrait};
                                   ^*******^

warn: Usage of deprecated feature `"deprecated-list-trait"` with no `#[feature("deprecated-list-trait")]` attribute. Note: "Use `starknet::storage::Vec`."
 --> /strkfarm-contracts/contracts/src/strategies/vesu_rebalance/vesu_rebalance.cairo:28:42
    use alexandria_storage::list::{List, ListTrait};
                                   ^*******^

    Finished `dev` profile target(s) in 31 seconds


Collected 43 test(s) from strkfarm_contracts package
Running 43 test(s) from src/
[PASS] strkfarm_contracts::helpers::Math::tests::test_min (gas: ~1)
[PASS] strkfarm_contracts::components::ekuboSwap::tests::test_ekubo_swap_slippage_check (gas: ~749)
[PASS] strkfarm_contracts::components::ekuboSwap::tests::test_ekubo_invalid_to_address (gas: ~210)
[PASS] strkfarm_contracts::components::ekuboSwap::tests::test_ekubo_invalid_from_address (gas: ~210)
[PASS] strkfarm_contracts::components::swap::test_swaps::test_max_slippage_same_tokens (gas: ~635)
[PASS] strkfarm_contracts::components::swap::test_swaps::test_max_slippage_diff_tokens_should_pass (gas: ~609)
[PASS] strkfarm_contracts::components::swap::test_swaps::test_max_slippage_diff_tokens_should_fail (gas: ~609)
[PASS] strkfarm_contracts::strategies::cl_vault::test::test_cl_vault::test_rebalance_invalid_permissions (gas: ~2200)
[PASS] strkfarm_contracts::helpers::pow::tests::test_ten_pow_overflow (gas: ~5)
[PASS] strkfarm_contracts::strategies::vesu_rebalance::test::test_vesu_rebalance::test_vesu_constructor (gas: ~2182)
[PASS] strkfarm_contracts::strategies::cl_vault::test::test_cl_vault::test_set_settings_invalid_permissions (gas: ~2194)
```

```
[PASS] strkfarm_contracts::strategies::cl_vault::test::test_cl_vault::test_set_settings_pass (gas: ~2192)
[PASS] strkfarm_contracts::strategies::cl_vault::test::test_cl_vault::test_set_incentives_invalid_permissions (gas: ~2193)
[PASS] strkfarm_contracts::components::ekuboSwap::tests::test_ekubo_swap_simple (gas: ~758)
[PASS] strkfarm_contracts::components::ekuboSwap::tests::test_ekubo_no_routes_err (gas: ~209)
[PASS] strkfarm_contracts::components::vesu::tests::test_vesu_component (gas: ~11603)
[PASS] strkfarm_contracts::helpers::safe_decimal_math::tests::test_mul_decimals_overflow (gas: ~5)
[PASS] strkfarm_contracts::helpers::safe_decimal_math::tests::test_mul_overflow (gas: ~2)
[PASS] strkfarm_contracts::helpers::safe_decimal_math::tests::test_div_decimals (gas: ~6)
[PASS] strkfarm_contracts::helpers::safe_decimal_math::tests::test_mul (gas: ~2)
[PASS] strkfarm_contracts::helpers::Math::tests::test_max (gas: ~1)
[PASS] strkfarm_contracts::helpers::safe_decimal_math::tests::test_mul_decimals (gas: ~6)
[PASS] strkfarm_contracts::strategies::vesu_rebalance::test::test_vesu_rebalance::test_vesu_deposit (gas: ~6667)
[PASS] strkfarm_contracts::helpers::pow::tests::test_ten_pow (gas: ~27)
[PASS] strkfarm_contracts::components::ekuboSwap::tests::test_ekubo_multihop_ekubo_swap (gas: ~933)
[PASS] strkfarm_contracts::strategies::vesu_rebalance::test::test_vesu_rebalance::test_vesu_withdraw (gas: ~11223)
[PASS] strkfarm_contracts::strategies::cl_vault::test::test_cl_vault::test_strk_xstrk_pool (gas: ~20163)
[PASS] strkfarm_contracts::strategies::vesu_rebalance::test::test_vesu_rebalance::test_vesu_harvest_and_withdraw (gas: ~14673)
[PASS] strkfarm_contracts::strategies::cl_vault::test::test_cl_vault::test_handle_ununsed_invalid_from_token (gas: ~2193)
[PASS] strkfarm_contracts::strategies::cl_vault::test::test_cl_vault::test_ekubo_withdraw (gas: ~4733)
[PASS] strkfarm_contracts::components::vesu::tests::test_hf_user (gas: ~841)
[PASS] strkfarm_contracts::helpers::safe_decimal_math::tests::test_div (gas: ~2)
[PASS] strkfarm_contracts::components::ekuboSwap::tests::test_ekubo_swap_exact_out (gas: ~835)
[PASS] strkfarm_contracts::strategies::cl_vault::test::test_cl_vault::test_handle_ununsed_invalid_to_token (gas: ~2194)
[PASS] strkfarm_contracts::strategies::cl_vault::test::test_cl_vault::test_ekubo_deposit (gas: ~5074)
[PASS] strkfarm_contracts::strategies::cl_vault::test::test_cl_vault::test_set_incentives_pass (gas: ~2125)
[PASS] strkfarm_contracts::strategies::cl_vault::test::test_cl_vault::test_handle_fees (gas: ~6256)
[PASS] strkfarm_contracts::strategies::cl_vault::test::test_cl_vault::test_clVault_constructer (gas: ~2221)
[PASS] strkfarm_contracts::strategies::cl_vault::test::test_cl_vault::test_harvest_cl_vault (gas: ~11046)
[PASS] strkfarm_contracts::strategies::cl_vault::test::test_cl_vault::test_ekubo_rebalance (gas: ~7152)
[PASS] strkfarm_contracts::strategies::vesu_rebalance::test::test_vesu_rebalance::test_vesu_rebalance_should_fail_relayer_role (gas: ~7073)
[PASS] strkfarm_contracts::strategies::vesu_rebalance::test::test_vesu_rebalance::test_vesu_rebalance_should_fail (gas: ~11605)
[PASS] strkfarm_contracts::strategies::vesu_rebalance::test::test_vesu_rebalance::test_vesu_rebalance_action (gas: ~12619)
Tests: 43 passed, 0 failed, 0 skipped, 0 ignored, 0 filtered out
```