CAIRO SECURITY CLAN

# STRKFARM

## SECURITY ASSESMENT REPORT

JULY 2024

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. 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

12 points of attention, where 1 is classified as Critical, 3 are classified as High, 3 are classified as Medium,1 is classified as Low,1 is classified as Informational and 3 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.

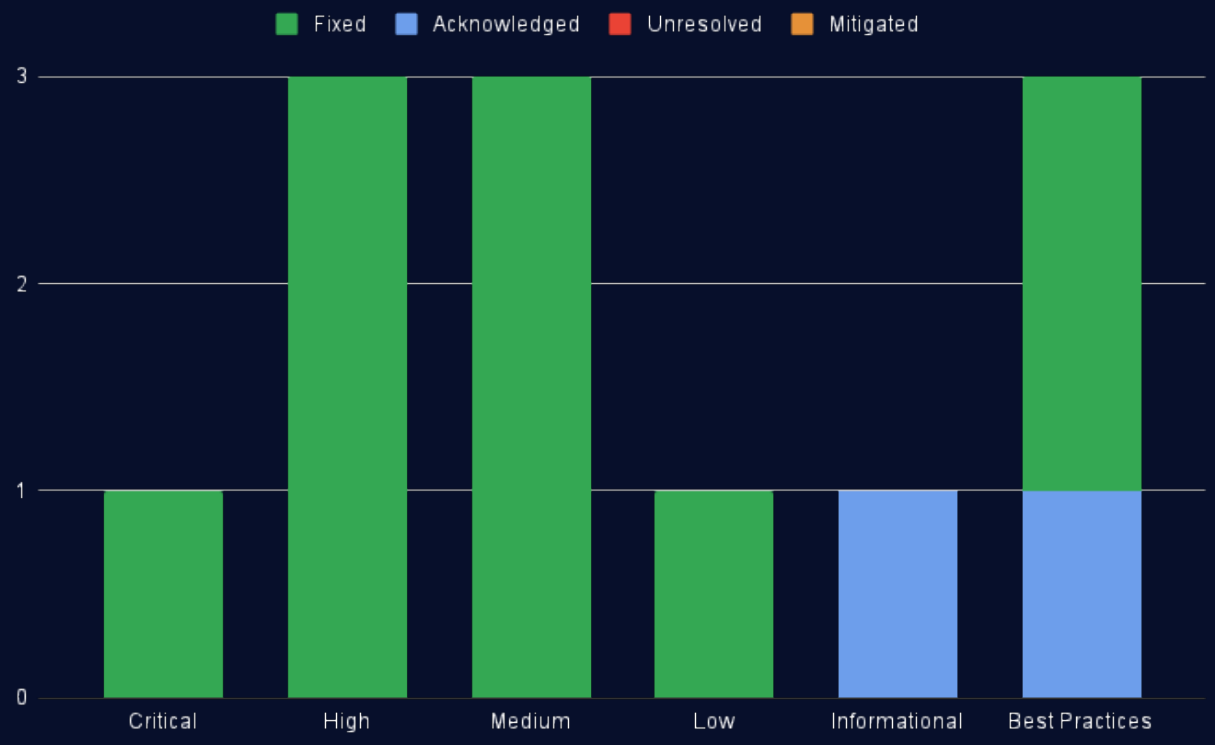


**Fig 1: Distribution of issues: Critical** (1), **High** (3), **Medium** (3), **Low** (1), **Informational** (1), **Best Practices** (3). **Distribution of status: Fixed** (10), **Acknowledged** (2), **Mitigated** (0), **Unresolved** (0).

# 4 Summary of Audit

| | |
|---|---|
| **Audit Type** | Security Review |
| **Cairo Version** | 2.6.3 |
| **Response from Client** | 23/08/2024 |
| **Final Report** | 26/08/2024 |
| **Repository** | strkfarm-core |
| **Initial Commit Hash** | 5a3d7710002b626b3690a043a91bf18a6e29627f |
| **Final Commit Hash** | 04dbbc4e7cc522e3596c6b3cdde1aecde673b201 |
| **Documentation** | Website documentation |
| **Test Suite Assessment** | Low |

## 4.1 Scoped Files

| | Contracts |
|---|---|
| 1 | src/components/ekubo.cairo |
| 2 | src/components/nostra.cairo |
| 3 | src/components/nostraSwap.cairo |
| 4 | src/components/swap.cairo |
| 5 | src/components/zkLend.cairo |
| 6 | src/components/common.cairo |
| 7 | src/components/harvester/defi_spring_default_style.cairo |
| 8 | src/components/harvester/defi_spring_ekubo_style.cairo |
| 9 | src/components/harvester/harvester_lib.cairo |
| 10 | src/components/harvester/interface.cairo |
| 11 | src/components/harvester/reward_shares.cairo |
| 12 | src/strats/delta_neutral_looping.cairo |
| 13 | src/strats/harvest_invest.cairo |
| 14 | src/external/pow.cairo |
| 14 | src/external/safe_decimal_math.cairo |
| 15 | src/helpers/ERC20Helper.cairo |
| 16 | src/helpers/constants.cairo |
| 17 | src/interfaces/lendcomp.cairo |

## 4.2 Issues

| | Findings | Severity | Update |
|---|---|---|---|
| 1 | Additional shares are accounted twice when users withdraw | Critical | Fixed |
| 2 | Swap routes can be used with unefficient routes | High | Fixed |
| 3 | Harvesting will always revert when `fee_percent > 0` | High | Fixed |
| 4 | Unminted shares are stuck when users withdraw during a round | High | Fixed |
| 5 | Missing slippage protection in function `_withdraw()` | Medium | Fixed |
| 6 | Unhandled edge case when `withdraw_amount2 > borrow1` during withdrawal | Medium | Fixed |
| 7 | Underlying to shares exchange rate could become invalid after rebalance | Medium | Fixed |
| 8 | Users cannot withdraw all from delta neutral looping | Low | Fixed |
| 9 | Function `rebalance()` should check activation condition using `min_health_factor` instead of `target_health_factor` | Informational | Acknowledged |
| 10 | Unnecessary dispatcher usage | Best Practices | Fixed |
| 11 | Unused codes | Best Practices | Fixed |
| 12 | Removing commented debug codes | 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;

b) **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 Critical

### 6.1.1 Additional shares are accounted twice when users withdraw

File(s): src/strats/delta_neutral_looping.cairo

**Description:** In the function `_withdraw()`, the position information of the user is fetched using `_describe_position()`. This function returns the deposit and borrow amounts, which are then used along with `position_percent_basis_points` to proceed with the withdrawal and repayment.

Next, the `save()` function is called to update the total shares of each action, and `_reduce_position()` is called to update the user's position. If `position_percent_basis_points = 100%`, it means the user has fully withdrawn their position.

At the end, there is additional logic related to `additional_shares` from the `reward_share` module. It gets the `additional_shares` entitled to the user and adds them to `position.acc1_supply_shares`.

```
let (position, position_description) = self._describe_position(
    self._address_to_u256(owner)
);
assert(position.acc1_supply_shares > 0, 'Withdraw::No Position');

// [...]

// withdraw zklend
zkLendDepositAction.withdraw(
    ref self,
    safe_decimal_math::div_round_down(
        position_description.deposit1 * position_percent_basis_points,
        10000
    )
);

// transfer remaining amount to receiver
let transfer_amount = zkLendDepositAction.total_reduce - nostraBorrowAction.total_reduce - require_main_token;
ERC20Helper::strict_transfer(zkLendDepositAction.token, receiver, transfer_amount);

// account reward shares
let all_shares = self.get_all_shares();
let total_zklend_deposit_shares_before = all_shares.acc1_supply_shares;
let (additional_shares, last_block, pending_round_points)
    = self.reward_share.get_additional_shares(receiver);

zkLendDepositAction.save(ref self);
zkLendBorrowAction.save(ref self);
nostraDepositAction.save(ref self);
nostraBorrowAction.save(ref self);

let token_id = self._address_to_u256(owner);
let mut position = self._reduce_position(
    token_id,
    zkLendDepositAction.current_shares,
    zkLendBorrowAction.current_shares,
    nostraDepositAction.current_shares,
    nostraBorrowAction.current_shares
);

let acc1_supply_shares = position.acc1_supply_shares;

position.acc1_supply_shares += additional_shares;
self.positions.write(token_id, position);
```

However, this is incorrect because the `additional_shares` are already accounted for in `_describe_position()` at the beginning of the function. As a result, users can withdraw the additional shares, but since they are added again to their positions, they can withdraw these shares again, causing a loss of funds for the pool.

```
fn _get_position(
    self: @ContractState,
    token_id: u256
) -> Position {
    let user = self._u256_to_address(token_id);
    let (additional_shares, _, _) = self.reward_share.get_additional_shares(user);
    self._get_position_given_additional_shares(token_id, additional_shares)
}
```

**Recommendation(s):** Consider not adding the `additional_shares` to users' positions again at the end of function `_withdraw()`.

**Status:**

**Update from client:** Fixed in commit.

## 6.2 High

### 6.2.1 Swap routes can be used with unefficient routes

File(s): src/strats/delta_neutral_looping.cairo

**Description:** The harvest function swaps the reward token into a base token, and then it will be used for re-investing (depending on strategy). However, the swap route can be passed as a parameter, and any user can pass low liquidity sources, which results in lower base tokens than expected.

```
fn harvest(
    ref self: ContractState,
    protocol1_rewards_contract: ContractAddress,
    claim1: Claim, proof1: Span<felt252>,
    protocol2_rewards_contract: ContractAddress,
    claim2: Claim, proof2: Span<felt252>,
    swapInfo: AvnuMultiRouteSwap
) {
    // ...
    config.double_harvest(
        ref self,
        ekuboSettings,
        claim1,
        proof1,
        snfSettings,
        claim2,
        proof2,
        swapInfo
    );
    self.renack.end();
}
```

**Recommendation(s):** Consider checking whether the swap route is the most efficient route or use constant routes.

**Status:** Fixed

**Update from client:** Fixed in commit.

### 6.2.2 Harvesting will always revert when `fee_percent > 0`

File(s): src/strats/delta_neutral_looping.cairo

**Description:** In the delta neutral looping strategy, the admin can configure `fee_percent` and `fee_receiver` to receive the fee when harvesting rewards. If `fee_percent > 0`, the function `after_update()` is called after harvesting to collect the fee and send it to `fee_receiver`. However, even though the fee is already deducted from the total amount and sent away, the `zkLendDepositAction` still calls the `deposit()` function with the full `amount`. As a result, the `deposit()` call will revert due to insufficient token balance.

```
fn after_update(ref self: ContractState, token: ContractAddress, amount: u256) {
    // ...
    let fee = (amount * self.fee_percent.read()) / 10000;
    if (fee > 0) {
        let receiver = self.fee_receiver.read();
        assert(receiver.is_non_zero(), 'Harvest::Invalid fee receiver');
        ERC20Helper::strict_transfer(token, receiver, fee);
    }
    let all_shares = self.get_all_shares();
    let (mut zkLendDepositAction, _, _, _) = self._get_actions();
    zkLendDepositAction.deposit(ref self, amount);
    let shares = zkLendDepositAction.convert_to_shares(@self, amount);
    self.reward_share.update_harvesting_rewards(amount,shares,all_shares.acc1_supply_shares);
}
```

**Recommendation(s):** Consider changing the deposit amount to `amount - fee` after sending the fee to the receiver.

**Status:** Fixed

**Update from client:** Fixed in commit.

### 6.2.3 Unminted shares are stuck when users withdraw during a round

File(s): src/components/harvester/reward_shares.cairo

**Description:** In the `reward_shares` module, the function `update_user_rewards()` is called every time a user deposits or withdraws. Besides updating the user's shares, it also updates the total round points of the current round. This is necessary because when users deposit or withdraw, the total shares of the pool change, which means the number of points per block changes.

```
// update total round shares
let mut rewards: RewardsInfo = self.rewards.read(current_index);
assert(rewards.amount == 0, 'Rewards already distributed');
let total_round_points = self.get_total_round_points(total_shares);
rewards.total_round_points += total_round_points;
rewards.block_number = get_block_number();
```

However, when users withdraw all of their funds and the current round is not finalized yet, the total round points are not updated. Even though users withdraw their shares during a round, their points in that round still contribute to the total points.

Additionally, due to the logic in the `get_additional_shares()` function, users cannot claim rewards after they have fully withdrawn. These rewards (unminted shares) will be stuck in the contract.

```
// no deposits by user
if (user_rewards.shares_owned == 0) {
  return (0, get_block_number(), 0);
}
```

**Recommendation(s):** Consider decreasing the total round points of the current round if users withdraw fully. Alternatively, allow users to claim their rewards even if they have already fully withdrawn.

**Status:** Fixed

**Update from client:** Fixed in commit.

## 6.3 Medium

### 6.3.1 Missing slippage protection in function `_withdraw()`

File(s): src/strats/delta_neutral_looping.cairo

**Description:** When users withdraw from the delta neutral looping strategy, the function takes a flash loan of the main token to repay the debt on Nostra. Then, it withdraws the secondary token from Nostra and uses it to repay the debt on ZkLend. If the withdrawn amount is insufficient, it tries to swap from the main token to cover the missing part. However, there is no slippage protection during the swap.

An attacker could manipulate the price of the Nostra pool before the withdrawal, causing `get_amounts_in()` to return an extremely high result and leading to losses for users trying to withdraw. The `swap()` function is also called with `min_amount_out = 0`, indicating missing slippage protection.

```
let mut require_main_token: u256 = 0;
if (required_excess_zkLend_repay_amount > 0) {
    /// println!("swapping");
    let amounts_out = self.nostra_swap_settings.read().get_amounts_in(
        required_excess_zkLend_repay_amount,
        array![zkLendDepositAction.token, zkLendBorrowAction.token],
    );

    require_main_token = *amounts_out.at(0);
    let borrow2 = safe_decimal_math::div_round_down(
        position_description.borrow2 * position_percent_basis_points,
        10000
    );
    let remaining_flash_amount = flashloan_amount - borrow2;
    assert(require_main_token <= remaining_flash_amount, 'Insuft flash amt');

    // swap
    self.nostra_swap_settings.read().swap(
        zkLendDepositAction.token,
        zkLendBorrowAction.token,
        require_main_token,
        0_u256,
        get_contract_address(),
    );
    /// println!("swapping done");
}
```

**Recommendation(s):** Consider allowing users to input the slippage percentage they are willing to accept if a swap occurs when calling the `withdraw()` function and use the price oracle to calculate the value for `min_amount_out`.

**Status:** Fixed

**Update from client:** Fixed in commit.

### 6.3.2 Unhandled edge case when `withdraw_amount2 > borrow1` during withdrawal

File(s): src/strats/delta_neutral_looping.cairo

**Description:** In the `_withdraw()` function, when the withdrawn amount from Nostra is not enough to repay the debt on ZkLend, the function will swap some main token to secondary token to cover the missing part.

```
let mut required_excess_zkLend_repay_amount: u256 = 0;
if (borrow1 > withdraw_amount2) {
    required_excess_zkLend_repay_amount = borrow1 - withdraw_amount2;
}
```

However, in the case where `withdraw_amount2 > borrow1`, the withdrawn amount from Nostra is more than the debt on ZkLend. The exceeding withdrawn amount from Nostra is not handled. After repaying the debt on ZkLend, the remaining secondary token is not transferred back to users or deposited back to ZkLend.

**Recommendation(s):** Consider swapping the remaining secondary token to main token and transferring it to the user when `withdraw_amount2 > borrow1` during withdrawal.

**Status:** Fixed

**Update from client:** Fixed in commit.

### 6.3.3 Underlying to shares exchange rate could become invalid after rebalance

File(s): src/strats/delta_neutral_looping.cairo

**Description:** In the delta neutral looping strategy, there are four actions, each handling one type of interaction with either ZkLend or Nostra protocols. Each action has its own accounting variables: total shares and total underlying. The exchange rate between these two values is calculated in the function `_update_exrate()` as

```
shares_to_underlying_ex_rate = all_shares * 10^27 / underlying / 10^token_offset

underlying_to_shares_ex_rate = underlying * 10^token_offset * 10^27 / all_shares
```

Now we look at the function `convert_to_shares()`. If `shares_to_underlying_ex_rate > 10^27`, it is considered an invalid exchange rate.

```
fn convert_to_shares(
    ref self: Action<TSettings>,
    state: @ContractState,
    amount: u256
) -> u256 {
    assert(self.shares_to_underlying_ex_rate > 0 && self.shares_to_underlying_ex_rate <= pow::ten_pow(27), '
     Invalid exrate [2]');
    safe_decimal_math::mul(amount * pow::ten_pow(self.token_offset), self.shares_to_underlying_ex_rate)
}

fn convert_to_underlying(
    self: @Action<TSettings>,
    state: @ContractState,
    shares: u256
) -> u256 {
    assert(*self.underlying_to_shares_ex_rate >= pow::ten_pow(27), 'Invalid exrate [1]');
    let underlying = safe_decimal_math::mul(shares, *self.underlying_to_shares_ex_rate);
    underlying / pow::ten_pow(*self.token_offset)
}
```

However, this invalid exchange rate could exist after a rebalance. Since the total shares won't change but the underlying amount (borrow/deposit amount) could decrease after rebalancing, `all_shares` could be larger than `underlying`. Assuming `token_offset == 0`, according to the formula in `_update_exrate()`, `shares_to_underlying_ex_rate` will be larger than `10^27` given `all_shares > underlying`.

The similar issue also exists in the function `convert_to_underlying()` for the `underlying_to_shares_ex_rate` variable.

**Recommendation(s):** Consider reviewing the check for the exchange rate in these 2 functions.

**Status:** Fixed

**Update from client:** Fixed in commit.

## 6.4 Low

### 6.4.1 Users cannot withdraw all from delta neutral looping

File(s): src/strats/delta_neutral_looping.cairo

**Description:** In `delta_neutral_looping.cairo`, users can withdraw by calling the function `withdraw()` and providing the `amount` input parameter. This amount is used to calculate `position_percent_basis_points` based on the estimated size of the current position.

```
fn withdraw(
    ref self: ContractState,
    amount: u256,
    receiver: ContractAddress
) {
    self.renack.start();
    self.common.assert_not_paused();

    let (_, desc) = self._describe_position(
        self._address_to_u256(get_caller_address())
    );
    assert(desc.estimated_size > 0, 'Withdraw::No Position');
    assert(desc.estimated_size >= amount, 'Withdraw::EXCESS_AMT');

    let position_percent_basis_points = (amount * 10000) / desc.estimated_size;
```

However, since the `estimated_size` of the user's position accounts for the interest accrued by the lending protocol, it keeps changing every second. Due to the nature of blockchain, users can't pre-calculate the position size at the moment their transactions are executed. As a result, when a user wants to withdraw all, they either send an `amount` larger than `estimated_size` and the transaction might revert, or the `amount` is less than `estimated_size` and they can only withdraw part of their position.

**Recommendation(s):** Consider adding functionality to allow users to withdraw all. For instance, when users pass in `amount = 2**256 - 1`, treat it as a request to withdraw all and set `position_percent_basis_points = 10000`.

**Status:** Fixed

**Update from client:** Fixed in commit.

## 6.5 Informational

### 6.5.1 Function `rebalance()` should check activation condition using `min_health_factor` instead of `target_health_factor`

File(s): src/strats/delta_neutral_looping.cairo

**Description:** In the delta neutral looping strategy, there are two configuration values related to the health factor: `min_health_factor` and `target_health_factor`, with the constraint `min_health_factor < target_health_factor`. The `min_health_factor` is used to check after each deposit/withdraw action, while `target_health_factor` is used as an activation condition for rebalance. This could cause the strategy to rebalance more than usual because any user can cause the health factor to be between these two values after a deposit/withdraw (`min_health_factor < hf < target_health_factor`). These operations are allowed, but it will activate a rebalance right after that because hf < `target_health_factor`.

```
fn rebalance(
    ...
) {
    self.renack.start();
    self.common.assert_not_paused();

    // dont allow rebalancing of healthy positions
    let (hf1, hf2) = self.health_factors();
    let targetHf = self.target_health_factor.read();
    assert(hf1 < targetHf || hf2 < targetHf, 'Rebalance::Already Healthy');
```

**Recommendation(s):** Consider checking health factors with `min_health_factor` instead of `target_health_factor`.

**Status:** Acknowledged

**Update from client:**

## 6.6 Best Practices

### 6.6.1 Unnecessary dispatcher usage

File(s): src/components/harvester/defi_spring_default_style.cairo

**Description:** Variable `_rewardToken` already set as constant. However, result struct gets token address from dispatcher.

```
let _rewardToken = constants::STRK_ADDRESS();
let rewardToken = ERC20ABIDispatcher {contract_address: _rewardToken};
// ...
ClaimResult {
  token: rewardToken.contract_address,
  amount
}
```

**Recommendation(s):** Consider using the constant address without getting from the dispatcher.

**Status:** Fixed

**Update from client:** Fixed in commit.

### 6.6.2 Unused codes

File(s): src/components/harvester/reward_shares.cairo, src/strats/delta_neutral_looping.cairo

**Description:** There are some unused codes left in the codebase.

```
// delta_neutral_looping.cairo
let temp = 55; // @audit unused

// reward_shares.cairo
pub struct Storage {
  // ...
  Ownable_owner: ContractAddress, // @audit unused
}

// harvest_invest.cairo
fn harvest(ref self: ContractState, claim: Claim, proof: Span<felt252>, swapInfo: AvnuMultiRouteSwap) {
    let settings = self.get_settings();
    let ekuboSettings = EkuboStyleClaimSettings {
        rewardsContract: settings.rewardsContract,
    };
    let config = HarvestConfig {};
    // todo: some build error, need to fix and uncomment
    // config.simple_harvest(
    //     ref self,
    //     ekuboSettings,
    //     claim,
    //     proof,
    //     swapInfo
    // );
}
```

**Recommendation(s):** Consider removing these unused codes to improve code quality.

**Status:** Fixed

**Update from client:** Fixed in commit.

### 6.6.3 Removing commented debug codes

File(s): src/

**Description:** The current codebase contains many console log codes used for debugging during development. Although these lines are commented out and do not affect code execution, it is recommended to remove them to improve code quality and make the overall codebase look cleaner.

```
// repay in nostra
nostraBorrowAction.repay(
    ref self,
    safe_decimal_math::div_round_down(
        position_description.borrow2 * position_percent_basis_points,
        10000
    )
);
/// println!("repay nostra done");

// withdraw from nostra
/// println!("nostra:: position_description.deposit2: {:?}", position_description.deposit2);
/// println!("nostra:: position_percent_basis_points: {:?}", position_percent_basis_points);
```

**Recommendation(s):** Prior to deployment, remove all commented debug statements to ensure a more polished and maintainable codebase.

**Status:** Acknowledged

**Update from client:**

# 7 Test Evaluation

## 7.1 Compilation Output

```
scarb build
    Updating git repository github.com/ekuboprotocol/abis
    Updating git repository github.com/zkdice-xyz/erc4626
    Updating git repository github.com/openzeppelin/cairo-contracts
    Updating git repository github.com/foundry-rs/starknet-foundry
   Compiling strkfarm v0.1.0 (\strkFarm\Scarb.toml)
warn: Unused variable. Consider ignoring by prefixing with `_`.
 --> \strkFarm\src\strats\harvest_invest.cairo:139:17
            let ekuboSettings = EkuboStyleClaimSettings {
                ^***********^

warn: Unused variable. Consider ignoring by prefixing with `_`.
 --> \strkFarm\src\strats\harvest_invest.cairo:142:17
            let config = HarvestConfig {};
                ^****^

warn: Unused variable. Consider ignoring by prefixing with `_`.
 --> \strkFarm\src\strats\delta_neutral_looping.cairo:1119:17
            let temp = 55;
                ^**^

    Finished release target(s) in 2 minutes
```

## 7.2 Tests Output

### 7.2.1 Cairo Tests

```
scarb test
     Running test strkfarm (snforge test --max-n-steps 30000000)
warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/components/zkLend.cairo:231:13
        let pre_deposit_amount = settings.deposit_amount(constants::USDC_ADDRESS(), this);
            ^****************^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/components/zkLend.cairo:233:13
        let post_deposit_amount = settings.deposit_amount(constants::USDC_ADDRESS(), this);
            ^*****************^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/components/zkLend.cairo:267:13
        let net_amount = settings.deposit_amount(constants::USDC_ADDRESS(), this);
            ^********^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/components/nostra.cairo:301:13
        let bal = ERC20Helper::balanceOf(constants::USDC_ADDRESS(), this);
            ^*^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/components/nostra.cairo:312:13
        let bal = ERC20Helper::balanceOf(constants::USDC_ADDRESS(), this);
            ^*^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/components/nostra.cairo:322:13
        let postbal = ERC20Helper::balanceOf(constants::USDC_ADDRESS(), this);
            ^*****^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/components/nostra.cairo:325:13
        let deposit_amt = nostraSettings.deposit_amount(constants::USDC_ADDRESS(), this);
            ^*********^
```

```
warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/components/nostra.cairo:328:13
        let borrow_amt = nostraSettings.borrow_amount(constants::USDC_ADDRESS(), this);
            ^********^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_delta_neutral.cairo:192:30
        let (_position_prev, description) = strategy.describe_position(_address_to_u256(user));
                             ^*********^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_delta_neutral.cairo:238:14
        let (hf1, hf2) = strategy.health_factors();
             ^*^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_delta_neutral.cairo:238:19
        let (hf1, hf2) = strategy.health_factors();
                  ^*^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_delta_neutral.cairo:265:25
        let (_position, description) = strategy.describe_position(_address_to_u256(user));
                        ^*********^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_delta_neutral.cairo:258:13
        let bal = ERC20Helper::balanceOf(strategy.config().main_token, this);
            ^*^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_delta_neutral.cairo:466:13
        let bal = ERC20Helper::balanceOf(constants::STRK_ADDRESS(), user);
            ^*^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_delta_neutral.cairo:498:13
        let bal = ERC20Helper::balanceOf(constants::USDC_ADDRESS(), user);
            ^*^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_delta_neutral.cairo:538:13
        let bal = ERC20Helper::balanceOf(constants::USDC_ADDRESS(), user);
            ^*^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:174:9
    let bal = ERC20Helper::balanceOf(constants::USDC_ADDRESS(), user);
        ^*^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:449:10
    let (posUser1, _) = state.describe_position(safe_decimal_math::address_to_u256(constants::TestUserUSDCLarge()
     ));
         ^******^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:397:10
    let (posUser2, _) = state.describe_position(safe_decimal_math::address_to_u256(user2));
         ^******^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:501:10
    let (posUser1, _) = state.describe_position(safe_decimal_math::address_to_u256(constants::TestUserUSDCLarge()
     ));
         ^******^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:609:10
```

```
    let (pos12, desc1) = state.describe_position(safe_decimal_math::address_to_u256(user1));
         ^***^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:609:17
    let (pos12, desc1) = state.describe_position(safe_decimal_math::address_to_u256(user1));
                ^***^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:610:10
    let (pos22, desc2) = state.describe_position(safe_decimal_math::address_to_u256(user2));
         ^***^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:610:17
    let (pos22, desc2) = state.describe_position(safe_decimal_math::address_to_u256(user2));
                ^***^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:735:9
    let reward = safe_decimal_math::fei_to_wei(50, 6);
        ^****^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:832:25
    let (expected_pos1, desc1) = state.describe_position(safe_decimal_math::address_to_u256(user1));
                        ^***^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:809:10
    let (pos1, _) = state.describe_position(safe_decimal_math::address_to_u256(user1));
         ^**^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:810:10
    let (pos2, _) = state.describe_position(safe_decimal_math::address_to_u256(user2));
         ^**^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:823:9
    let exected_shares = reward * total_acc1_shares / zUSDCBal;
        ^************^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:834:10
    let (expected_pos2, desc2) = state.describe_position(safe_decimal_math::address_to_u256(user2));
         ^***********^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:834:25
    let (expected_pos2, desc2) = state.describe_position(safe_decimal_math::address_to_u256(user2));
                        ^***^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:843:10
    let (pos12, desc1) = state.describe_position(safe_decimal_math::address_to_u256(user1));
         ^***^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:850:9
    let bal_pre = ERC20Helper::balanceOf(constants::USDC_ADDRESS(), user1);
        ^*****^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_unit_delta_neutral.cairo:852:9
    let bal_post = ERC20Helper::balanceOf(constants::USDC_ADDRESS(), user1);
        ^******^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_reward_shares.cairo:314:10
```

```
    let (user1_shares, _, _) = state.get_additional_shares(mock_user1);
         ^**********^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_reward_shares.cairo:315:10
    let (user2_shares, _, _) = state.get_additional_shares(mock_user2);
         ^**********^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_reward_shares.cairo:338:10
    let (user1_shares, _, _) = state.get_additional_shares(mock_user1);
         ^**********^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_reward_shares.cairo:339:10
    let (user2_shares, _, _) = state.get_additional_shares(mock_user2);
         ^**********^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_reward_shares.cairo:361:10
    let (user1_shares, _, _) = state.get_additional_shares(mock_user1);
         ^**********^

warning: Unused variable. Consider ignoring by prefixing with `_`.
 --> /STRKFARM/src/tests/test_reward_shares.cairo:362:10
    let (user2_shares, _, _) = state.get_additional_shares(mock_user2);
         ^**********^

   Compiling strkfarm v0.1.0 (/STRKFARM/Scarb.toml)
    Finished release target(s) in 25 seconds
[WARNING] RPC node with the url https://starknet-mainnet.public.blastapi.io uses incompatible version 0.6.0.
     Expected version: 0.7.0


[PASS] strkfarm::external::safe_decimal_math::tests::test_div (gas: ~2)
[PASS] strkfarm::external::pow::tests::test_ten_pow (gas: ~5)
[PASS] strkfarm::external::safe_decimal_math::tests::test_div_decimals (gas: ~2)
[PASS] strkfarm::tests::test_reward_shares::test_reward_shares::test_two_users_deposit_harvest_random (gas:
    ~1838)
[PASS] strkfarm::external::safe_decimal_math::tests::test_mul_overflow (gas: ~2)
[PASS] strkfarm::external::pow::tests::test_ten_pow_overflow (gas: ~2)
[PASS] strkfarm::tests::test_nimbora::tests::test_master_constructer (gas: ~360)
[PASS] strkfarm::tests::test_auto_strk::tests::test_upgrade_should_panic (gas: ~752)
[PASS] strkfarm::external::safe_decimal_math::tests::test_mul_decimals_overflow (gas: ~2)
[PASS] strkfarm::tests::test_auto_strk::tests::test_constructor (gas: ~766)
[PASS] strkfarm::tests::test_reward_shares::test_reward_shares::test_rewards_deposit_harvest (gas: ~797)
[PASS] strkfarm::tests::test_auto_strk::tests::test_set_settings_should_panic (gas: ~752)
[PASS] strkfarm::tests::test_nimbora::tests::test_deposit_assetInvalid (gas: ~1180)
[PASS] strkfarm::tests::test_nimbora::tests::test_deposit_tokenInvalid (gas: ~1453)
[PASS] strkfarm::components::swap::test_swaps::test_max_slippage_same_tokens (gas: ~623)
[PASS] strkfarm::tests::test_reward_shares::test_reward_shares::test_two_users_deposit_harvest_boundary (gas:
    ~1082)
[PASS] strkfarm::tests::test_reward_shares::test_reward_shares::test_rewards_deposit_harvest_harvest (gas: ~1073)
[PASS] strkfarm::components::zkLend::tests::test_zklend_component (gas: ~1666)
[PASS] strkfarm::components::swap::test_swaps::test_max_slippage_diff_tokens_should_fail (gas: ~596)
[PASS] strkfarm::tests::test_auto_strk::tests::test_strk_sow_withdraw (gas: ~2395)
[PASS] strkfarm::tests::test_reward_shares::test_reward_shares::test_deposit_withdraw_harvest_redeem (gas: ~1068)
[PASS] strkfarm::components::swap::test_swaps::test_max_slippage_diff_tokens_should_pass (gas: ~596)
[PASS] strkfarm::tests::test_unit_delta_neutral::tests_unit_delta_neutral_mm::test_deposit_zero_amt (gas: ~2387)
[PASS] strkfarm::tests::test_unit_delta_neutral::tests_unit_delta_neutral_mm::test_deposit_zero_receiver (gas:
    ~2387)
[PASS] strkfarm::tests::test_auto_strk::tests::test_usdc_deposit_withdraw (gas: ~4633)
[PASS] strkfarm::tests::test_auto_strk::tests::test_strk_deposit_withdraw (gas: ~1998)
[PASS] strkfarm::tests::test_unit_delta_neutral::tests_unit_delta_neutral_mm::test_unit_deposit_withdraw (gas:
    ~16912)
[PASS] strkfarm::components::nostra::tests::test_nostra_component (gas: ~2770)
[PASS] strkfarm::tests::test_delta_neutral::tests_delta_neutral_mm::test_deposit_withdraw_no_timediff (gas:
    ~29771)
[PASS] strkfarm::tests::test_unit_delta_neutral::tests_unit_delta_neutral_mm::test_first_deposit (gas: ~10192)
```

```
[PASS] strkfarm::tests::test_unit_delta_neutral::tests_unit_delta_neutral_mm::
    test_first_deposit_new_user_deposit_second_first_user_deposit_new_user_deposit_again (gas: ~30681)
[PASS] strkfarm::external::safe_decimal_math::tests::test_mul_decimals (gas: ~2)
[PASS] strkfarm::external::safe_decimal_math::tests::test_mul (gas: ~2)
[PASS] strkfarm::tests::test_unit_delta_neutral::tests_unit_delta_neutral_mm::test_deposit_harvest_deposit (gas:
    ~27580)
[PASS] strkfarm::tests::test_delta_neutral::tests_delta_neutral_mm::test_transfer_fail (gas: ~17147)
[PASS] strkfarm::tests::test_delta_neutral::tests_delta_neutral_mm::
    test_deposit_withdraw_with_timediff_one_user_50p (gas: ~36619)
[PASS] strkfarm::tests::test_unit_delta_neutral::tests_unit_delta_neutral_mm::test_unit_deposit_harvest_withdraw
    (gas: ~28007)
[PASS] strkfarm::tests::test_delta_neutral::tests_delta_neutral_mm::
    test_deposit_withdraw_with_timediff_one_user_100p (gas: ~37858)
[PASS] strkfarm::tests::test_delta_neutral::tests_delta_neutral_mm::
    test_strk_deposit_withdraw_with_timediff_multi_users (gas: ~55324)
[PASS] strkfarm::tests::test_auto_strk::tests::test_usdc_sow_withdraw (gas: ~3765)
[PASS] strkfarm::tests::test_delta_neutral::tests_delta_neutral_mm::
    test_deposit_withdraw_with_timediff_multi_users (gas: ~63343)
[PASS] strkfarm::tests::test_unit_delta_neutral::tests_unit_delta_neutral_mm::test_second_deposit (gas: ~17882)
[PASS] strkfarm::tests::test_delta_neutral::tests_delta_neutral_mm::test_dnmm_usdc_harvest (gas: ~19725)
[PASS] strkfarm::tests::test_delta_neutral::tests_delta_neutral_mm::test_strk_deposit_withdraw_no_timediff (gas:
    ~25177)
Tests: 44 passed, 0 failed, 0 skipped, 0 ignored, 0 filtered out

Latest block number = 671812 for url = https://starknet-mainnet.public.blastapi.io
```