Cyberscope

Audit Report

Contrax

June 2024

Repository	https://github.com/Contrax-co/contrax-smart-contracts/tree/main
Commit	120dbc5021f9197ace8db1e0b6b4c6f5d47db5f1
Audited by	© cyberscope

Table of Contents

Review

Repository	https://github.com/Contrax-co/contrax-smart-contracts/tree/main
Commit	120dbc5021f9197ace8db1e0b6b4c6f5d47db5f1

Audit Updates

Initial Audit	12 Feb 2024 https://github.com/cyberscope-io/audits/blob/main/contrax/v1/audit.pdf
Corrected Phase 2	19 Feb 2024 https://github.com/cyberscope-io/audits/blob/main/contrax/v2/audit.pdf
Corrected Phase 3	21 Feb 2024 https://github.com/cyberscope-io/audits/blob/main/contrax/v3/audit.pdf
Corrected Phase 4	27 Jun 2024

Source Files

Filename	SHA256
strategies/steer/strategy-steer.sol	5192c44a1583cd52acabd3206dc7b657e02b5b9b90feacbb06fa59fe262b8dc4
strategies/steer/strategy-steer-weth-sushi.sol	88a3a5c809d7c280889ea0961a89cfdc1b49b1eec19b65c904a95e104dcc0ab9
strategies/steer/strategy-steer-usdc-usdce.sol	b843b6498add0ff3246fa596c15e88db55d7db6711aaa573eb125441ee5598b2

strategies/steer/strategy-steer-base.sol	a096025545c595dba4a53a204db8d69f4883c1c955620f791904a2f16e5f9880
controllers/steer-controller.sol	72ebfa76fb5d0b3ee319d4020131f451c688f3f41231e3ea53304652589604ad
Utils/PriceCalculatorV3.sol	9be98e7ff43e487fbb455a734c2bc6a769b83da8003b3bcc15b33da06c668211

Overview

Audit Scope

The contract audit scope includes the following contracts:

- Controllers: steer-controller.sol
- Utils: PriceCalculatorV3.sol
- Steer: strategy-steer-base.sol, strategy-steer-usdc-usdce.sol, strategy-steer-weth-sushi.sol, strategy-steer.sol

SteerController

The SteerController contract is a comprehensive management contract for coordinating vaults and strategies within a decentralized finance (DeFi) ecosystem. It manages mappings that associate tokens with their respective vaults and strategies, ensuring streamlined operations. Key functionalities include setting and approving vaults and strategies, transferring funds between vaults and strategies, handling emergency withdrawals, and ensuring proper fund allocation. By maintaining a robust and flexible framework, SteerController enhances the efficiency and security of DeFi strategies, allowing for dynamic adjustments and secure fund management under defined governance rules.

PriceCalculatorV3 Contract

The PriceCalculatorV3 contract extends the functionalities of its predecessor to support Uniswap V3 pools, providing a more sophisticated approach to price calculations. It includes mechanisms for setting pool fees, adding stable tokens, and calculating token prices in USD using Uniswap V3's tick data. The contract uses the OracleLibrary to fetch price quotes at specific ticks, ensuring precision in token and ETH price calculations. Additionally, it maintains governance control, allowing only authorized addresses to modify critical parameters, thus enhancing the robustness and flexibility of DeFi strategies.

StrategySteerBase Contract

The StrategySteerBase contract extends the functionalities of the StrategySteer contract, providing a framework for managing liquidity positions and executing swaps within Uniswap

V3. It includes a comprehensive harvest function that collects rewards, converts them into the appropriate tokens, and reinvests them into the Steer Vault. This contract ensures precise token swaps through an abstract _swap function, allowing derived contracts to implement specific swap logic. Additionally, it manages approvals and deposits to the Steer Periphery contract, ensuring efficient and secure reinvestment of assets, and calculates token prices and ratios for accurate reward handling.

StrategySteerUsdcUsdce Contract

The StrategySteerUsdcUsdce contract inherits from StrategySteerBase, specifically tailored for managing the USDC-USDC.e pair. It implements the _swap function to perform token swaps using the Uniswap V3 router, facilitating efficient conversions between USDC and USDC.e. This contract leverages the existing infrastructure provided by StrategySteerBase to manage rewards, perform swaps, and reinvest assets, ensuring optimal returns for liquidity providers while maintaining compatibility with Uniswap V3's fee structure and liquidity management capabilities.

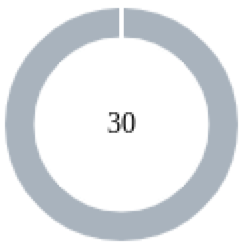
StrategySteerWethSushi Contract

The StrategySteerWethSushi contract, derived from StrategySteerBase, is designed to manage the WETH-SUSHI liquidity pair. It utilizes the SushiSwap V2 router to execute swaps, with specific routing logic to handle SUSHI's liquidity primarily through WETH. The contract implements the _swap function to facilitate token conversions, ensuring efficient and accurate swaps between WETH and SUSHI. By building on StrategySteerBase, it inherits robust reward management and reinvestment functionalities, optimizing returns for liquidity providers while integrating seamlessly with the SushiSwap ecosystem.

StrategySteer Contract

The StrategySteer contract serves as the foundation for advanced DeFi strategies, integrating with the PriceCalculatorV3 for precise pricing mechanisms. It manages a variety of parameters, including performance and withdrawal fees, and supports the addition of harvesters for reward collection. The contract defines the harvest function to collect and convert rewards into the want tokens, which are then reinvested into the strategy. With functionalities for setting and managing pool fees, whitelisting harvesters, and handling emergency operations through delegate calls, StrategySteer provides a versatile and secure framework for implementing complex DeFi strategies.

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	30

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	0	30	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	ACE	Access Control Enhancement	Acknowledged
●	CR	Code Repetition	Acknowledged
●	CCR	Contract Centralization Risk	Acknowledged
●	DPI	Decimals Precision Inconsistency	Acknowledged
●	EIS	Excessively Integer Size	Acknowledged
●	IDI	Immutable Declaration Improvement	Acknowledged
●	IEM	Inconsistent Error Messages	Acknowledged
●	MCC	Missing Constructor Checks	Acknowledged
●	MEM	Missing Error Messages	Acknowledged
●	MEE	Missing Events Emission	Acknowledged
●	MSV	Missing Strategy Validation	Acknowledged
●	MSPV	Missing Swap Pair Verification	Acknowledged
●	MU	Modifiers Usage	Acknowledged
●	ODM	Oracle Decimal Mismatch	Acknowledged

●	PAV	Pair Address Validation	Acknowledged
●	PBV	Percentage Boundaries Validation	Acknowledged
●	PLPI	Potential Liquidity Provision Inadequacy	Acknowledged
●	PLO	Precision Loss Optimization	Acknowledged
●	RSML	Redundant SafeMath Library	Acknowledged
●	RC	Repetitive Calculations	Acknowledged
●	TSI	Tokens Sufficiency Insurance	Acknowledged
●	L02	State Variables could be Declared Constant	Acknowledged
●	L04	Conformance to Solidity Naming Conventions	Acknowledged
●	L06	Missing Events Access Control	Acknowledged
●	L11	Unnecessary Boolean equality	Acknowledged
●	L13	Divide before Multiply Operation	Acknowledged
●	L14	Uninitialized Variables in Local Scope	Acknowledged
●	L16	Validate Variable Setters	Acknowledged
●	L17	Usage of Solidity Assembly	Acknowledged
●	L18	Multiple Pragma Directives	Acknowledged
●	L19	Stable Compiler Version	Acknowledged

ACE - Access Control Enhancement

Criticality	Minor / Informative
Location	strategies/steer/strategy-steer.sol#L87,98
Status	Unresolved

Description

The contract is currently using individual `require` statements for access control in the `whitelistHarvester` function. This approach can lead to redundant code and potential inconsistencies in access control logic. The `onlyBenevolent` modifier, which checks if the caller is either a harvester, governance, or strategist, could be used to streamline and standardize access control across functions that require similar permissions.

```
 modifier onlyBenevolent() {
     require(harvesters[msg.sender] || msg.sender == governance
|| msg.sender == strategist);
     _;
   }

   function whitelistHarvester(address _harvester) external {
     require(msg.sender == governance || msg.sender ==
strategist || harvesters[msg.sender], "not authorized");
     harvesters[_harvester] = true;
   }
```

Recommendation

It is recommended to use the `onlyBenevolent` modifier in the `whitelistHarvester` function to ensure consistent access control and reduce redundancy. This will enhance code readability and maintainability while ensuring that only authorized addresses can execute the function.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	Utils/PriceCalculatorV3.sol#L116,121,126,131 Utils/PriceCalculatorV3.sol#L42 strategies/steer/strategy-steer.sol#L253
Status	Acknowledged

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

Specifically, the authorized addresses have the ability to execute critical functions that can withdraw all tokens, handle tokens stuck in strategies, transfer tokens directly, set the pool fee, and invoke the `execute` function using a delegatecall to a `_target` contract.

```
 function withdrawAll(address _token) public {
     require(msg.sender == strategist || msg.sender ==
governance, "!strategist");
     IStrategy(strategies[_token]).withdrawAll();
   }

  function inCaseTokensGetStuck(address _token, uint256
_amount) public {
     require(msg.sender == strategist || msg.sender ==
governance, "!governance");
     IERC20(_token).safeTransfer(msg.sender, _amount);
   }

  function inCaseStrategyTokenGetStuck(address _strategy,
address _token) public {
     require(msg.sender == strategist || msg.sender ==
governance, "!governance");
     IStrategy(_strategy).withdraw(_token);
   }

  function withdraw(address _token, uint256 _amount) public {
     require(msg.sender == vaults[_token], "!vault");
     IStrategy(strategies[_token]).withdraw(_amount);
   }
```

```
 function setPoolFees(uint24 _fee) external onlyGovernance {
     poolsFee.push(_fee);
   }
```

```
 function execute(address _target, bytes memory _data) public
payable returns (bytes memory response) {
     require(msg.sender == timelock, "!timelock");
     require(_target != address(0), "!target");

     // call contract in current context
     assembly {
       let succeeded := delegatecall(sub(gas(), 5000), _target,
add(_data, 0x20), ...
       }
     }
   }
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Location	Utils/PriceCalculatorV3.sol#L62,71,93 strategies/steer/strategy-steer-base.sol#L14,46
Status	Acknowledged

Description

The decimals field of a contract's ERC20 token can be used to specify the number of decimal places that the token uses. For example, if decimals are set to `8`, it means that the smallest unit of the token is `0.00000001`, and if decimals are set to `18`, it means that the smallest unit of the token is `0.000000000000000001`.

However, there is an inconsistency in the way that the decimals field is handled in some ERC20 contracts. The ERC20 specification does not specify how the decimals field should be implemented, and as a result, some contracts use different precision numbers.

This inconsistency can cause problems when interacting with these contracts, as it is not always clear how the decimals field should be interpreted. For example, if a contract expects the decimals field to be 18 digits, but the contract being interacted with uses 8 digits, the result of the interaction may not be what was expected.

Aditionally, the `minimumAmount` varaible is not set to have decimals, while is compared with the `_reward` varaible which includes decimals.

```
uint256 lpPriceInEth = (lpPriceInWei * PRECISION) / 1e18;
...
1e18, // fixed point to 18 decimals
...
return (PriceFromOracle * PRECISION) / 1e6;
```

```
uint256 public constant minimumAmount = 1000;
...
require(_reward >= minimumAmount, "Insignificant input
amount");
```

Recommendation

To avoid these issues, it is important to carefully review the implementation of the decimals field of the underlying tokens. The team is advised to normalize each decimal to one single source of truth. A recommended way is to scale all the decimals to the greatest token's decimal. Hence, the contract will not lose precision in the calculations.

The following example depicts 3 tokens with different decimals precision.

ERC20	Decimals
Token 1	6
Token 2	9
Token 3	18

All the decimals could be normalized to 18 since it represents the ERC20 token with the greatest digits.

EIS - Excessively Integer Size

Criticality	Minor / Informative
Location	strategies/steer/strategy-steer.sol#L34,37,41
Status	Acknowledged

Description

The contract is using a bigger unsigned integer data type that the maximum size that is required. By using an unsigned integer data type larger than necessary, the smart contract consumes more storage space and requires additional computational resources for calculations and operations involving these variables. This can result in higher transaction costs, longer execution times, and potential scalability bottlenecks.

Given that the maximum values for `withdrawalTreasuryFee` , `withdrawalDevFundFee` , and `keepReward` are 100,000 and 10,000 respectively, a `uint16` data type could be used instead of `uint256` . This would optimize storage usage and potentially reduce gas costs, as uint16 can store values up to 65,535, which is sufficient for these limits.

```
uint256 public withdrawalTreasuryFee = 0;
...
uint256 public withdrawalDevFundFee = 0;
...
uint256 public keepReward = 1000;
```

Recommendation

To address the inefficiency associated with using an oversized unsigned integer data type, it is recommended to accurately determine the required size based on the range of values the variable needs to represent. It is recommended to use a `uint16` data type.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	PriceCalculatorV3.sol#L38 Utils/PriceCalculatorV3.sol#L38
Status	Acknowledged

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
governance
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

IEM - Inconsistent Error Messages

Criticality	Minor / Informative
Location	controllers/steer-controller.sol#L117,122
Status	Acknowledged

Description

The contract is using identical `require` checks within different functions but provides different error messages for these checks. This inconsistency in error messages may lead to confusion and difficulty in diagnosing issues when the contract encounters an error condition. Specifically, the same condition (`require(msg.sender == strategist || msg.sender == governance)`) returns the error message `"!strategist"` in one function and `"!governance"` in another.

```
require(msg.sender == strategist || msg.sender == governance,
"!strategist");
...
require(msg.sender == strategist || msg.sender == governance,
"!governance");
```

Recommendation

It is recommended to standardize error messages to accurately reflect the conditions that trigger them. This will ensure clarity and consistency, making it easier for users and developers to understand and debug the contract's behavior.

MCC - Missing Constructor Checks

Criticality	Minor / Informative
Location	controllers/steer-controller.sol#L36
Status	Acknowledged

Description

The contract is currently designed to initialize with external addresses for `governance`, `strategist`, `timelock`, `devfund` and `treasury` parameters within its constructor. However, it lacks validation checks to ensure that these addresses are not the zero address (0x0). The absence of such validation exposes the contract to potential risks, as initializing crucial components like the governance, strategist, or timelock with the zero address could lead to malfunctioning of the contract, hindering operations such as governance decisions, and access control management. This oversight may result in a scenario where the contract is deployed in an unusable state or susceptible to security vulnerabilities, affecting the overall integrity and functionality of the contract.

```
 constructor(address _governance, address _strategist, address
_timelock, address _devfund, address _treasury) {
    governance = _governance;
    strategist = _strategist;
    timelock = _timelock;
    devfund = _devfund;
    treasury = _treasury;
  }
```

Recommendation

It is recommended to implement checks within the constructor to validate that the addresses for the `governance`, `strategist`, `timelock`, `devfund` and `treasury` are not the zero address before setting them. This can be achieved by adding require statements for each parameter to ensure they are not equal to the zero address. Such a preventative measure will enhance the contract's security and robustness by ensuring that all critical components are properly initialized, thereby preventing potential operational failures or security loopholes. This validation should be considered a standard practice in smart contract development to safeguard against common pitfalls associated with improper initialization.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	strategy-steer.sol#L59,60,61,62,63,81 PriceCalculatorV3.sol#L37 strategies/steer/strategy-steer.sol#L59,60,61,62,63,81 Utils/PriceCalculatorV3.sol#L37
Status	Acknowledged

Description

The contract is missing error messages. Specifically, there are no error messages to accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(_want != address(0))
require(_governance != address(0))
require(_strategist != address(0))
require(_controller != address(0))
require(_timelock != address(0))
require(harvesters[msg.sender] || msg.sender == governance ||
msg.sender == strategist)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	controllers/steer-controller.sol#L56,66,71 PriceCalculatorV3.sol#L43 strategies/steer/strategy-steer.sol#L186
Status	Acknowledged

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
strategist = _strategist;
governance = _governance;
timelock = _timelock;
...
```

```
stableTokens.push(_stableTokens);
```

```
 function setPoolFees(address _token0, address _token1, uint24
_poolFee) external onlyGovernance {
    require(_poolFee > 0, "pool fee must be greater than 0");
    require(_token0 != address(0) && _token1 != address(0),
"invalid address");

    poolFees[_token0][_token1] = _poolFee;
    // populate mapping in the reverse direction, deliberate
choice to avoid the cost of comparing addresses
    poolFees[_token1][_token0] = _poolFee;
  }
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

MSV - Missing Strategy Validation

Criticality	Minor / Informative
Location	controllers/steer-controller.sol#L111
Status	Acknowledged

Description

The contract is designed to facilitate the redirection of funds to investment strategies via the `earn` function, which takes `_token` and `_amount` as parameters. This function locates the corresponding strategy for the given `_token` but fails to validate whether the retrieved `_strategy` address actually exists or is valid. Without this check, if a strategy for the specified `_token` does not exist within the strategies mapping, the function will proceed with operations on an undefined or zero address, leading to potential failure of the transaction or unintended behavior.

```
function earn(address _token, uint256 _amount) public {
    address _strategy = strategies[_token];
    IERC20(_token).safeTransfer(_strategy, _amount);
}
```

Recommendation

It is recommended to incorporate a validation check immediately after retrieving the `_strategy` address from the strategies mapping to ensure it is not the zero address. This can be achieved with a `require` statement asserting that `_strategy` is a valid contract address, coupled with a clear error message if the condition is not met. This preventative measure ensures that only defined and active strategies are interacted with, minimizing the risk of errors or lost funds.

MSPV - Missing Swap Pair Verification

Criticality	Minor / Informative
Location	strategies/steer/strategy-steer-base.sol#L49 strategy-steer.sol#L113
Status	Acknowledged

Description

The contract is designed to facilitate the swapping of a rewards token with another token, utilizing the `_swap` function to execute this exchange. This functionality is triggered under certain conditions, particularly when the rewards token does not match predefined token pairs (token0 or token1). However, the contract currently lacks a validation step since it does not verify whether a valid trading pair exists for these tokens before performing the swap. This oversight can lead to unsuccessful swaps if the pair does not exist on the trading platform, potentially causing transaction failures and leading to increased gas costs for users, besides potential disruptions in the intended utility of the rewards system.

```
if (rewardToken != token0 && rewardToken != token1) {
  _swap(rewardToken, token0, tokenInAmount0);
  _swap(rewardToken, token1, tokenInAmount1);
...
  function setRewardToken(address _rewardToken) external {
    require(msg.sender == timelock || msg.sender == strategist,
"!timelock");
    rewardToken = _rewardToken;
  }
```

Recommendation

It is recommended to add an additional checks in the function where the rewards token is exchanged to verify that the token pair exists. This can be implemented by querying the trading platform's existing pair registry or through a direct call to ensure the pair's validity before executing the swap. Implementing this validation will enhance the contract's reliability by preventing attempts to swap non-existent pairs, thus safeguarding user transactions and optimizing the contract's performance.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	controllers/steer-controller.sol#L45,50,55,60,65,70,76,81,86,91,97,117 strategies/steer/strategy-steer.sol#L104,109,124,129,134,129,144,149,154,159,180,189,206,217
Status	Acknowledged

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(msg.sender == governance, "!governance");
require(msg.sender == timelock, "!timelock");
require(msg.sender == strategist || msg.sender == governance,
"!strategist");
```

```
require(msg.sender == timelock, "!timelock");
...
require(msg.sender == governance, "!governance");
...
require(msg.sender == controller, "!controller");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

ODM - Oracle Decimal Mismatch

Criticality	Minor / Informative
Location	Utils/PriceCalculatorV3.sol#L50,77,86
Status	Acknowledged

Description

The contract relies on data retrieved from an external Oracle to make critical calculations. However, the contract does not include a verification step to align the decimal precision of the retrieved data with the precision expected by the contract's internal calculations. This mismatch in decimal precision can introduce substantial errors in calculations involving decimal values.

```
 function getPriceInTermsOfToken1(int24 tick) public pure
returns (uint256 priceU18) {
    priceU18 = OracleLibrary.getQuoteAtTick(
      tick,
      1e18, // fixed point to 18 decimals
      address(1), // since we want the price in terms of
token0/token1
      address(0)
    );
  }

  function calculateTokenPriceInUsd(address _token, address
_pairAddress) public view returns (uint256) {
    ...
    uint256 lpPriceInWei;
    if (_token == token0) {
      lpPriceInWei = getPriceInTermsOfToken0(tick);
    } else {
      lpPriceInWei = getPriceInTermsOfToken1(tick);
    }
    uint256 lpPriceInEth = (lpPriceInWei * PRECISION) / 1e18;
    uint256 ethPriceInUsd = calculateEthPriceInUsdc();
    ethPriceInUsd = ethPriceInUsd / PRECISION;
    return lpPriceInEth * ethPriceInUsd;
  }

  function calculateEthPriceInUsdc() public view returns
(uint256) {
    IUniswapV3Pool pool = IUniswapV3Pool(weth_Usdc_Pool_V3);
    (, int24 tick, , , , , ) = pool.slot0();

    uint256 PriceFromOracle = getPriceInTermsOfToken0(tick);

    //removing usdc decimals
    return (PriceFromOracle * PRECISION) / 1e6;
  }
```

Recommendation

The team is advised to retrieve the decimals precision from the Oracle API in order to proceed with the appropriate adjustments to the internal decimals representation.

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	strategies/steer/strategy-steer-weth-sushi.sol#L24
Status	Acknowledged

Description

The contract is missing address validation in the pair address argument. The absence of validation reveals a potential vulnerability, as it lacks proper checks to ensure the integrity and validity of the pair address provided as an argument. The pair address is a parameter used in certain methods of decentralized exchanges for functions like token swaps and liquidity provisions.

The absence of address validation in the pair address argument can introduce security risks and potential attacks. Without proper validation, if the owner's address is compromised, the contract may lead to unexpected behavior like loss of funds.

Specifically the `tokenIn` address which can represent a reward token can be changed.

```
 function _swap(address tokenIn, address tokenOut, uint256
amountIn) internal override {
    address[] memory path;

    // sushi only has liquidity with eth, so always route with
weth to swap sushi
    if (tokenIn != weth && tokenOut != weth && (tokenIn ==
sushi || tokenOut == sushi)) {
        path = new address[](3);
        path[0] = tokenIn;
        path[1] = weth;
        path[2] = tokenOut;
        ...
        }
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pair address argument, it is recommended to implement comprehensive address validation mechanisms.

A recommended approach could be to verify pair existence in the decentralized application. Prior to interacting with the pair address contract, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by querying the provider's contract or utilizing external libraries that provide contract verification services.

PBV - Percentage Boundaries Validation

Criticality	Minor / Informative
Location	strategies/steer/strategy-steer.sol#L108,118,123,128
Status	Acknowledged

Description

The contract utilizes variables for percentage-based calculations that are required for its operations. These variables are involved in multiplication and division operations to determine proportions related to the contract's logic. If such variables are set to values beyond their logical or intended maximum limits, it could result in incorrect calculations. This misconfiguration has the potential to cause unintended behavior or financial discrepancies, affecting the contract's integrity and the accuracy of its calculations.

```
 function setKeepReward(uint256 _keepReward) external {
    require(msg.sender == timelock, "!timelock");
    keepReward = _keepReward;
  }

  function setFeeDistributor(address _feeDistributor) external
{
    require(msg.sender == governance, "not authorized");
    feeDistributor = _feeDistributor;
  }

  function setWithdrawalDevFundFee(uint256
_withdrawalDevFundFee) external {
    require(msg.sender == timelock, "!timelock");
    withdrawalDevFundFee = _withdrawalDevFundFee;
  }

  function setWithdrawalTreasuryFee(uint256
_withdrawalTreasuryFee) external {
    require(msg.sender == timelock, "!timelock");
    withdrawalTreasuryFee = _withdrawalTreasuryFee;
  }
```

Recommendation

To mitigate risks associated with boundary violations, it is important to implement validation checks for variables used in percentage-based calculations. Ensure that these variables do not exceed their maximum logical values. This can be accomplished by incorporating `require` statements or similar validation mechanisms whenever such variables are assigned or modified. These safeguards will enforce correct operational boundaries, preserving the contract's intended functionality and preventing computational errors.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	strategies/steer/strategy-steer-weth-sushi.sol#L43
Status	Acknowledged

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
UniswapRouterV2(router).swapExactTokensForTokens(amountIn, 0,
path, address(this), block.timestamp);
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

PLO - Precision Loss Optimization

Criticality	Minor / Informative
Location	Utils/PriceCalculatorV3.sol#L62
Status	Acknowledged

Description

The contract contains multiple sequential calculations involving division and multiplication by constants. As a result, it is experiencing issues with precision loss due to these operations. Specifically, the computation of `lpPriceInEth` and `ethPriceInUsd` is done separately with intermediate divisions and multiplications that can lead to inaccuracies. This issue arises because `lpPriceInEth` is calculated by converting `lpPriceInWei` using a precision factor, and then `ethPriceInUsd` is adjusted by dividing again by the precision factor. These intermediate steps introduce rounding errors that affect the final calculation of `lpPriceInUsd`.

```
uint256 lpPriceInEth = (lpPriceInWei * PRECISION) / 1e18;
uint256 ethPriceInUsd = calculateEthPriceInUsdc();
ethPriceInUsd = ethPriceInUsd / PRECISION;
return lpPriceInEth * ethPriceInUsd;
```

Recommendation

It is recommended to simplify and streamline the calculation by maintaining precision throughout the entire process. The calculation should be adjusted to first determine `ethPriceInUsd` accurately and then directly compute the value of `lpPriceInUsd` without intermediate divisions. By multiplying `lpPriceInWei` directly with `ethPriceInUsd` and then dividing by `1e18`, we can avoid unnecessary precision loss and ensure the final value is accurate. This method reduces the risk of rounding errors and improves the overall precision of the contract’s financial calculations.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	strategy-steer.sol strategy-steer-base.sol steer-controller.sol strategies/steer/strategy-steer.sol strategies/steer/strategy-steer-base.sol controllers/steer-controller.sol
Status	Acknowledged

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	strategies/steer/strategy-steer-base.sol#L92,168
Status	Acknowledged

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive.

Specifically, the `steerVaultTokens` method is called multiple times.

```
 function calculateSteerVaultTokensRatio(uint256 _amountIn)
internal returns (uint256, uint256) {
    (address token0, address token1) = steerVaultTokens();
    (uint256 amount0, uint256 amount1) = getTotalAmounts();
    (uint256 token0Price, uint256 token1Price) =
calculateSteerVaultTokensPrices();
    ...
    }

function calculateSteerVaultTokensPrices() internal returns
(uint256 token0Price, uint256 token1Price) {
    (address token0, address token1) = steerVaultTokens();
    ...}
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	strategies/steer/strategy-steer-base.sol#L29
Status	Acknowledged

Description

The `rewardToken` tokens are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks.

```
 function harvest() public override onlyBenevolent {
     require(rewardToken != address(0), "!rewardToken");
     uint256 _reward =
IERC20(rewardToken).balanceOf(address(this));
     require(_reward > 0, "!reward");
     uint256 _keepReward = _reward.mul(keepReward).div(keepMax);

IERC20(rewardToken).safeTransfer(IController(controller).treasu
ry(), _keepReward);
     ...
     }
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the tokens within the contract itself. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Utils/PriceCalculatorV3.sol#L26,28 strategy-steer.sol#L21 strategy-steer-weth-sushi.sol#L20,21 strategy-steer-usdc-usdce.sol#L19 strategy-steer-base.sol#L13 PriceCalculatorV3.sol#L26,28
Status	Acknowledged

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
address public weth =
0x82aF49447D8a07e3bd95BD0d56f35241523fBab1
address public weth_Usdc_Pool_V3 =
0xC6962004f452bE9203591991D15f6b388e09E8D0
address public steerPeriphery =
0x806c2240793b3738000fcb62C66BF462764B903F
address public router =
0x1b02dA8Cb0d097eB8D57A175b88c7D8b47997506
address public sushi =
0xd4d42F0b6DEF4CE0383636770eF773390d85c61A
address public router =
0xE592427A0AEce92De3Edee1F18E0157C05861564
address public uniV3Factory =
0x1F98431c8aD98523631AE4a59f267346ea31F984
address public weth_Usdc_Pair =

0x905dfCD5649217c42684f23958568e533C711Aa3
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Utils/PriceCalculatorV3.sol#L28,41,46,50 strategy-steer.sol#L91,96,103,108,113,118,123,128,133,138,143,148,153,158,169,179,188,205,253 strategy-steer-base.sol#L70,140,162 steer-controller.sol#L44,49,54,59,64,69,75,80,85,90,96,107,111,116,121,126,131 PriceCalculatorV3.sol#L28,41,46,50
Status	Acknowledged

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
address public weth_Usdc_Pool_V3 =
0xC6962004f452bE9203591991D15f6b388e09E8D0
uint24 _fee
address _stableTokens
address _token
address _pairAddress
address _harvester
uint256 _keep
uint256 _keepReward
address _rewardToken
address _feeDistributor
uint256 _withdrawalDevFundFee
uint256 _withdrawalTreasuryFee
uint256 _performanceDevFee
uint256 _performanceTreasuryFee

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	strategy-steer.sol#L145,150
Status	Acknowledged

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
strategist = _strategist
governance = _governance
```

Recommendation

To avoid this issue, it's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	steer-controller.sol#L98
Status	Acknowledged

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(approvedStrategies[_token][_strategy] == true,
"!approved")
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Utils/PriceCalculatorV3.sol#L62,64,65 strategy-steer-base.sol#L167,171 PriceCalculatorV3.sol#L62,64,65
Status	Acknowledged

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 lpPriceInEth = (lpPriceInWei * PRECISION) / 1e18
ethPriceInUsd = ethPriceInUsd / PRECISION
return lpPriceInEth * ethPriceInUsd

uint256 token0Value = ((token0Price * amount0) / (10 **
uint256(IERC20(token0).decimals())))
uint256 token0Amount = (_amountIn * token0Value) / totalValue
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	strategy-steer.sol#L239
Status	Acknowledged

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 i
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	strategy-steer.sol#L115,120,145,150,155,160 steer-controller.sol#L37,38,39,40,41,46,51,56,61,66
Status	Acknowledged

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
rewardToken = _rewardToken
feeDistributor = _feeDistributor
strategist = _strategist
governance = _governance
timelock = _timelock
controller = _controller
devfund = _devfund
treasury = _treasury
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	strategy-steer.sol#L258
Status	Acknowledged

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
      let succeeded := delegatecall(sub(gas(), 5000), _target,
add(_data, 0x20), mload(_data), 0, 0)
      let size := returndatasize()

      response := mload(0x40)
      mstore(0x40, add(response, and(add(add(size, 0x20),
0x1f), not(0x1f))))
...

      switch iszero(succeeded)
      case 1 {
        // throw if delegatecall failed
        revert(add(response, 0x20), size)
      }
    }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	steer-controller.sol#L3,4 controllers/steer-controller.sol#L3,4
Status	Acknowledged

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity 0.8.4;
pragma experimental ABIEncoderV2;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Utils/PriceCalculatorV3.sol#L3 PriceCalculatorV3.sol#L3
Status	Acknowledged

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
StrategySteer	Implementation	PriceCalculatorV3		
		Public	✓	PriceCalculatorV3
	balanceOf	Public		-
	whitelistHarvester	External	✓	-
	revokeHarvester	External	✓	-
	setKeep	External	✓	-
	setKeepReward	External	✓	-
	setRewardToken	External	✓	-
	setFeeDistributor	External	✓	-
	setWithdrawalDevFundFee	External	✓	-
	setWithdrawalTreasuryFee	External	✓	-
	setPerformanceDevFee	External	✓	-
	setPerformanceTreasuryFee	External	✓	-
	setStrategist	External	✓	-
	setGovernance	External	✓	-
	setTimelock	External	✓	-
	setController	External	✓	-
	getPoolFee	Public		-

	setPoolFees	External	✓	onlyGovernance
	withdraw	External	✓	-
	withdraw	External	✓	-
	withdrawForSwap	External	✓	-
	withdrawAll	External	✓	-
	harvest	Public	✓	-
	depositToSteerVault	Internal	✓	
	getTotalAmounts	Public		-
	steerVaultTokens	Public		-
	_returnAssets	Internal	✓	
	_approveTokenIfNeeded	Internal	✓	
	execute	Public	Payable	-
StrategySteerWethSushi	Implementation	StrategySteerBase		
		Public	✓	StrategySteerBase
	_swap	Internal	✓	
StrategySteerUsdcUsdce	Implementation	StrategySteerBase		
		Public	✓	StrategySteerBase
	_swap	Internal	✓	
StrategySteerBase	Implementation	StrategySteer		

		Public	✓	StrategySteer
	_swap	Internal	✓	
	harvest	Public	✓	onlyBenevolent
	depositToSteerVault	Internal	✓	
	calculateSteerVaultTokensPrices	Internal	✓	
	isStableToken	Internal		
	getPrice	Internal	✓	
	fetchPool	Internal	✓	
	calculateSteerVaultTokensRatio	Internal	✓	
SteerController	Implementation			
		Public	✓	-
	setDevFund	Public	✓	-
	setTreasury	Public	✓	-
	setStrategist	Public	✓	-
	setGovernance	Public	✓	-
	setTimelock	Public	✓	-
	setVault	Public	✓	-
	approveVaultConverter	Public	✓	-
	revokeVaultConverter	Public	✓	-
	approveStrategy	Public	✓	-
	revokeStrategy	Public	✓	-
	setStrategy	Public	✓	-

	balanceOf	External		-
	earn	Public	✓	-
	withdrawAll	Public	✓	-
	inCaseTokensGetStuck	Public	✓	-
	inCaseStrategyTokenGetStuck	Public	✓	-
	withdraw	Public	✓	-
PriceCalculator V3	Implementation			
		Public	✓	-
	setPoolFees	External	✓	onlyGovernance
	setStableTokens	External	✓	onlyGovernance
	calculateTokenPriceInUsd	Public		-
	getPriceInTermsOfToken0	Public		-
	getPriceInTermsOfToken1	Public		-
	calculateEthPriceInUsdc	Public		-

Summary

The Contrax contract is designed to operate within the decentralized finance (DeFi) ecosystem, embodying principles of decentralization and financial innovation. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io