Cyberscope

Audit Report

BeraTrax

April 2025

Files Vault, ArberaZapper, InfraredZapper, KodiakZapper, SteerZapper, ZapperBase, LpRouter, SwapRouter, ArberaStrategy, InfraredStrategy, KodiakStrategy, SteerStrategy, SInfraredFactory, StrategyBase, ArberaFactory, KodiakFactory, SteerFactory, StrategyFactoryBase, VaultFactory, ControllerFactory, Controller

Audited by © cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Audit Updates

Initial Audit	17 Jan 2025 https://github.com/cyberscope-io/audits/blob/main/c5-btx/v1/audit.pdf
Corrected Phase 2	31 Jan 2025 https://github.com/cyberscope-io/audits/blob/main/c5-btx/v2/audit.pdf
Corrected Phase 3	09 Mar 2025 https://github.com/cyberscope-io/audits/blob/main/c5-btx/v3/audit.pdf
Corrected Phase 4	28 Mar 2025 https://github.com/cyberscope-io/audits/blob/main/c5-btx/v4/audit.pdf
Corrected Phase 5	21 Apr 2025

Source Files

Filename	SHA256
controllers/Controller.sol	1f60969ef7e143e6bfe716eed5918e89a3967d4e33f6574d76fb683abebd7d9f
factories/ArberaFactory.sol	0c006dcb7009a2ffade587696b917c87bdbcf4cb2431162bddf34a984dbf43d4
factories/ControllerFactory.sol	ba96b16cea05b58457ebbe8f0ee62eeb29cae8373aeff086af89c86e2bdb5253
factories/InfraredFactory.sol	fc95098f4ff35437101f5ab2fd8da58bffac1de26497afce345f15ef5dfc8995

factories/KodiakFactory.sol	930ae8ec9002eb2a4f24fbf6a1a3b16cd7c6f1342ba9c1a5b3ffb4c8a87298af
factories/SteerFactory.sol	259af16121589e77868c64bcea34331eb2722fcc2178291af3c0be0f9efffa07
factories/StrategyFactoryBase.sol	e5830234f43c16b95be70f338cba980b2aacfeae777657ea2bf651db5998ac15
factories/VaultFactory.sol	cdf805ce49d75b3f821bd434dbacc586f5eda3d93d5b16d73e7ce3787c963d2a
strategies/ArberaStrategy.sol	ec66fb891b874e3efc37f2599b80cd230fd0ca5d155fa601276ae85bc2248099
strategies/InfraredStrategy.sol	5cf42969420bd4fda4d029e34093339d9dff9db7e986cd428c7a8772a54a52e1
strategies/KodiakStrategy.sol	91edb466b8ed717a4e9b8a8d2280f8cbc2f888687778ae82fbaa370a61cf7590
strategies/SteerStrategy.sol	d643f75eef8f1d477cc18a2f3723c335ba529c1831ae5a4bb620b4a736ebae2b
strategies/StrategyBase.sol	91b69707fd7df73bedcd539e6e844e5b3ff59a5032f1f6fa409dd6b2720779fa
utils/LpRouter.sol	34db14c4e8cba6eb0ef004ec71abc31fcbc52d4d5deb5b4a0681828eda21dac3
utils/SwapRouter.sol	c59aec113feca63393760199398e39761fbf2468e12046c11655d2d2b0aa5479
vaults/Vault.sol	b6e8a55ff242d83a1812ae38a45f100b2f5f8650a3f7b97eb6167f96c66d8b3a
zappers/ArberaZapper.sol	e981c1db24e1e9243e1009f2f29f3fb4c8f0480d929cb03ca650a3175c3162cd
zappers/InfraredZapper.sol	49ed3484421b714f13ed933a719abfcf69d62ef200111e8e284c8686f05204c7
zappers/KodiakZapper.sol	b2e49fbbdada45d2ffa8d9399a683f2abddb0489ad8cae30ee89b310aec3181f
zappers/SteerZapper.sol	60df68b94a165ba9e17bfdac13721677de028eef1b4a1012368282bfd704a91c
zappers/ZapperBase.sol	d4958fd65bf68284863f838009300defba835d5811c5fb80692149e4c4bf90b6

Overview

The contracts implement a modular and efficient yield aggregator built on the Ethereum Virtual Machine (EVM). Contrax optimises yield generation by utilising a combination of smart contracts that automate asset management and maximise returns. The dApp provides users with a seamless platform to deposit their assets and earn competitive APR. Its architecture is designed for scalability, security, and flexibility, ensuring that users can confidently participate in an efficient and transparent ecosystem for yield farming.

Controller

The Controller serves as the central management layer connecting the Vaults to their respective investment Strategies. Its primary purpose is to oversee the flow of funds from the Vaults to the Strategies and ensure that assets are optimally allocated to generate yield. The Controller maintains control over approved Strategies, allowing governance to update or revoke them as needed to adapt to changing market conditions. By acting as an intermediary, the Controller provides a secure and modular architecture, separating user deposits from the underlying investment logic while enforcing robust access controls and operational flexibility.

Factories

The Factory is responsible for the deployment and initialization of Vaults and Controllers, enabling a seamless creation process for these core components. By standardizing the deployment of Vault-Controller pairs, the Factory ensures consistency in configuration and governance integration. It allows for scalability and adaptability by enabling developers and strategists to deploy new Vaults and Controllers for different assets, while ensuring the system adheres to predefined rules and relationships. As a central hub for new deployments, the Factory streamlines the expansion of the protocol while maintaining security and governance integrity.

Strategies

The Strategies are the yield-generating engines of the system, implementing specific logic for investing assets into staking pools, liquidity farms, or other financial products. Their goal is to maximize returns on assets deposited by the Vaults while adhering to risk and

operational constraints defined by governance. Each Strategy is tailored to a specific investment opportunity and can be updated or replaced as market conditions evolve. By abstracting investment logic, Strategies provide flexibility and modularity, allowing the protocol to adapt quickly to new yield sources without impacting the user-facing components of the system.

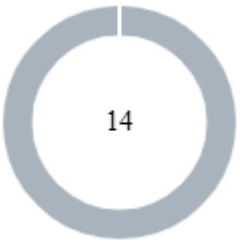
Vault

The Vault is the user-facing contract that manages deposits, withdrawals, and the representation of user stakes through Vault shares. When users deposit assets into the Vault, they receive shares that represent their proportional claim to the total assets under management. The Vault works closely with the Controller to allocate idle assets to Strategies through the `earn` function, ensuring that deposits are continuously put to productive use. During withdrawals, the Vault redeems shares for assets, either using its own reserves or retrieving funds from the Controller. By acting as a secure intermediary, the Vault simplifies user interactions while managing the complexities of yield generation.

Zappers

The Zappers are designed to enhance user convenience by automating asset conversions and liquidity provision for deposits and withdrawals. They allow users to interact with Vaults using various tokens, handling the necessary swaps, liquidity additions, and token approvals behind the scenes. Zappers streamline the process of entering and exiting Vaults, eliminating the need for users to manually manage token conversions or intermediate steps. By abstracting away these complexities, Zappers improve accessibility and usability, enabling a broader range of users to participate in the protocol’s yield-generating ecosystem.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	14

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	14	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	UTPD	Unverified Third Party Dependencies	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	ELFM	Exceeds Fees Limit	Unresolved
●	MN	Misspelled Naming	Unresolved
●	MC	Missing Check	Unresolved
●	UBUFL	Unintended Balance Used For Liquidity	Unresolved
●	SAU	Swapped Amount Uninitialized	Unresolved
●	ORA	Overwriting Rewards Amount	Unresolved
●	GRLNS	Gamma Remove Liquidity Not Supported	Unresolved
●	PRE	Potential Reentrance Exploit	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

UTPD - Unverified Third Party Dependencies

Criticality	Minor / Informative
Location	strategies/StrategyBase.sol#L191 strategies/InfraredStrategy.sol#L79 utils/LpRouter.sol#L107,133
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```
function _swapBGTToAsset() internal returns (uint256 amount) {
  uint256 balance = IERC20(bgt).balanceOf(address(this));
  if (balance > 0) {
    bgt.redeem(address(this), balance);
    ...
    }
  ...
}
```

```
function deposit() public override nonReentrant {
  ...
  staking.stake(balance);
}
```

Additionally, in `addLiquidity` and `removeLiquidity`, the `lp` can be added by the users. The contract performs operations based on the outcomes of the `lp's` functions which could also produce security issues.

```
address lp
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	utils/SwapRouter.sol#L461,531,579,621,655,690
Status	Unresolved

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.

```
function _swapWithRoute(
  address tokenIn,
  address tokenOut,
  uint256 amountIn,
  uint256 amountOutMinimum,
  address recipient
) internal returns (uint256 amountOut)

function _swapBex(
  address tokenIn,
  address tokenOut,
  uint256 amountIn,
  uint256 amountOutMinimum,
  address recipient,
  IBeraPool pool
) internal returns (uint256 amountOut)

function _swapV3(
  address tokenIn,
  address tokenOut,
  uint256 amountIn,
  uint256 amountOutMinimum,
  address recipient,
  address router,
  address factory
) internal returns (uint256 amountOut)

function _swapV3WithPath(
  address[] memory path,
  uint256 amountIn,
  uint256 amountOutMinimum,
  address recipient,
  address router,
  address factory
) internal returns (uint256 amountOut)

function _swapV2(
  address tokenIn,
  address tokenOut,
  uint256 amountIn,
  uint256 amountOutMinimum,
  address recipient,
  address router
) internal returns (uint256 amountOut)

function _swapV2WithPath(
  address[] memory path,
  uint256 amountIn,
  uint256 amountOutMinimum,
  address recipient,
```

```
    address router
) internal returns (uint256 amountOut)
```

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.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	controllers/Controller.sol#L135,144,153,162,171,181,191,200,210,245,253,261 vaults/Vault.sol#L139,150,161,172,179,201,240 utils/LpRouter.sol#L75,83,92,101 utils/SwapRouter.sol#L126,134,142,151,160,169,183 factories/ControllerFactory.sol#L59,71,88 factories/VaultFactory.sol#L55,67,85 factories/StrategyFactoryBase.sol#L65,74,83,224 factories/ArberaFactory.sol#L56 factories/InfraredFactory.sol#L88 factories/KodiakFactory.sol#L56 factories/SteerFactory.sol#L56 strategies/StrategyBase.sol#L203,211,219,228,237,246,255,264,273,282,291,300,309,334,349,357,385,397,419 strategies/InfraredStrategy.sol#L101,145,150 strategies/ArberaStrategy.sol#L88,127 zappers/ZapperBase.sol#L116,126,137,148,158,169,180
Status	Unresolved

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.

```
function setDevFund(address devfundAddress) public
onlyGovernance
function setTreasury(address treasuryAddress) public
onlyGovernance
function setStrategist(address strategistAddress) public
onlyGovernance
function setGovernance(address governanceAddress) public
onlyGovernance
function setTimelock(address timelockAddress) public
onlyTimelock
function setVault(address asset, address vault) public
onlyStrategist
function approveStrategy(address asset, address strategy)
public onlyTimelock
function revokeStrategy(address asset, address strategy) public
onlyGovernance
function setStrategy(address asset, address strategy) public
nonReentrant onlyStrategist
function withdrawAll(address asset) public nonReentrant
onlyStrategist
function inCaseTokensGetStuck(address asset, uint256 amount)
public onlyGovernance
function inCaseStrategyTokenGetStuck(address strategy, address
asset) public onlyGovernance
```

```
function setMin(uint256 minRatio) external onlyGovernance
function setDepositFee(uint16 newFee) external onlyGovernance
function setWithdrawFee(uint16 newFee) external onlyGovernance
function setGovernance(address governanceAddress) external
onlyGovernance
function setFeeRecipient(address feeRecipientAddress) external
onlyGovernance
function setTimelock(address timelockAddress) external
onlyTimelock
function setController(address controllerAddress) external
onlyTimelock
function harvest(address token, uint256 amount) external
nonReentrant onlyController
```

```
function setGovernance(address governanceAddress) public
onlyGovernance
function setSwapRouter(ISwapRouter swapRouterAddress) external
onlyGovernance
function setBexLpToTokenIn(address bexLp, address tokenIn)
external onlyGovernance
function setRouter(uint8 dex, address router) external
onlyGovernance
```

```
function setGovernance(address governanceAddress) public
onlyGovernance
function setDefaultDex(uint8 dex) external onlyGovernance
function setRouter(uint8 dex, address router) external
onlyGovernance
function setFactory(uint8 dex, address factory) external
onlyGovernance
function setDexFeeDenominator(uint8 dex, uint256
feeDenominator) external onlyGovernance
function setPool(address tokenIn, address tokenOut, address
pool) external onlyGovernance
function setSwapRoute(address tokenIn, address tokenOut,
SwapRoutePath[] memory path, bool reversePath) external
onlyGovernance
```

```
function setDev(address devAddress) external onlyDev
function setWhitelistedStrategyFactory(address factoryAddress,
bool isWhitelisted) external onlyDev
function createController(address asset, address
governanceAddress, address strategistAddress, address
timelockAddress, address devfundAddress, address
treasuryAddress) external onlyWhitelistedStrategyFactory
returns (IController controller)
```

```
function setDev(address devAddress) external onlyDev
function setWhitelistedStrategyFactory(address factoryAddress,
bool isWhitelisted) external onlyDev
function createVault(address asset, address governanceAddress,
address strategistAddress, address timelockAddress, address
devfundAddress, address treasuryAddress, address
controllerAddress, uint16 initialDepositFee, uint16
initialWithdrawFee) external onlyWhitelistedStrategyFactory
returns (IVault vault)
```

```
function setDev(address devAddress) external onlyDev
function setVaultFactory(address factoryAddress) external
onlyDev
function setControllerFactory(address factoryAddress) external
onlyDev
function createVault(VaultCreateParams calldata params)
external onlyDev onlyNewAsset(params.asset) returns (IVault
vault, IController controller, IStrategy strategy)
```

```
function createVaultWithParams(VaultParams calldata params)
external virtual onlyDev onlyNewAsset(params.asset) returns
(IVault vault, IController controller, IStrategy strategy)
```

```
function whitelistHarvester(address harvesterAddress) external
onlyBenevolent
function revokeHarvester(address harvesterAddress) external
onlyBenevolent
function setWithdrawalDevFundFee(uint16 fee) external
onlyTimelock
function setWithdrawalTreasuryFee(uint16 fee) external
onlyTimelock
function setPerformanceDevFee(uint16 fee) external onlyTimelock
function setPerformanceTreasuryFee(uint16 fee) external
onlyTimelock
function setStrategist(address strategistAddress) external
onlyGovernance
function setGovernance(address governanceAddress) external
onlyGovernance
function setTimelock(address timelockAddress) external
onlyTimelock
function setController(address controllerAddress) external
onlyTimelock
function setSwapRouter(address swapRouterAddress) external
onlyGovernance
function setLpRouter(address lpRouterAddress) external
onlyGovernance
function setZapper(address zapperAddress) external
onlyGovernance
function harvest() public virtual nonReentrant onlyBenevolent
function withdraw(address token) external nonReentrant
onlyController returns (uint256 balance)
function withdraw(uint256 amount) external nonReentrant
onlyController
function withdrawForSwap(uint256 amount) external nonReentrant
onlyController returns (uint256 balance)
function withdrawAll() external nonReentrant onlyController
returns (uint256 balance)
function execute(address target, bytes memory data) public
payable onlyTimelock returns (bytes memory response)
```

```
function harvest() public override onlyBenevolent
function setRewardTokensLength(uint256 newRewardTokensLength)
external onlyGovernance
function setStaking(address newStaking) external onlyGovernance
```

```
function harvest() public override onlyBenevolent
function setRewardToken(address _rewardToken) external
onlyGovernance
```

```
function setGovernance(address governanceAddress) external
onlyGovernance
function setSwapRouter(address routerAddress) external
onlyGovernance
function setLpRouter(address routerAddress) external
onlyGovernance
function setStableCoin(address stablecoinAddress) external
onlyGovernance
function setZapInFee(uint16 newFee) external onlyGovernance
function setZapOutFee(uint16 newFee) external onlyGovernance
function setFeeRecipient(address newRecipient) external
onlyGovernance
```

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.

ELFM - Exceeds Fees Limit

Criticality	Minor / Informative
Location	strategies/StrategyBase.sol#L26,237,246
Status	Unresolved

Description

The contract timelock address has the authority to increase performance fees over the allowed limit of 25%. The timelock address may take advantage of it by calling the `setPerformanceDevFee` or `setPerformanceTreasuryFee` function with a high percentage value.

```
uint16 public constant MAX_PERFORMANCE_FEE = 5000;
function setPerformanceDevFee(uint16 fee) external onlyTimelock
{
    if (fee + performanceTreasuryFee > MAX_PERFORMANCE_FEE)
revert FeeTooHigh(fee, MAX_PERFORMANCE_FEE);
    uint16 old = performanceDevFee;
    performanceDevFee = fee;
    emit PerformanceDevFeeChanged(old, fee);
}
function setPerformanceTreasuryFee(uint16 fee) external
onlyTimelock {
    if (fee + performanceDevFee > MAX_PERFORMANCE_FEE) revert
FeeTooHigh(fee, MAX_PERFORMANCE_FEE);
    uint16 old = performanceTreasuryFee;
    performanceTreasuryFee = fee;
    emit PerformanceTreasuryFeeChanged(old, fee);
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. The team should carefully manage the private keys of the timelock's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

MN - Misspelled Naming

Criticality	Minor / Informative
Location	utils/SwapRouter.sol#L183
Status	Unresolved

Description

The contract is designed to manage swap routes through the `setSwapRoute` function. However, within this function, the word "route" is used instead of "router". This misspelling may cause confusion for developers and users interacting with the code, as it deviates from the expected terminology used to describe swap routes. While this does not directly affect the functionality of the contract, it may reduce readability and increase the likelihood of misunderstandings or mistakes during future development or maintenance.

```
function setSwapRoute(
  address tokenIn,
  address tokenOut,
  SwapRoutePath[] memory path,
  bool reversePath
) external onlyGovernance {
  SwapRoutePath[] storage swapRouteForward =
swapRoutes[tokenIn][tokenOut];
  if (swapRouteForward.length > 0) delete
swapRoutes[tokenIn][tokenOut];
  for (uint256 i = 0; i < path.length; i++) {
    swapRouteForward.push(path[i]);
    if (!path[i].isMultiPath && path[i].pool != address(0)) {
      pools[path[i].tokenIn][path[i].tokenOut] = path[i].pool;
      pools[path[i].tokenOut][path[i].tokenIn] = path[i].pool;
    }
  }
  if (reversePath) {
    SwapRoutePath[] storage swapRouteReverse =
swapRoutes[tokenOut][tokenIn];
    if (swapRouteReverse.length > 0) delete
swapRoutes[tokenOut][tokenIn];
    for (uint256 i = 0; i < path.length; i++) {
      SwapRoutePath memory inversePath = path[path.length - i -
1];
      swapRouteReverse.push(
        SwapRoutePath({
          tokenIn: inversePath.tokenOut,
          tokenOut: inversePath.tokenIn,
          dex: inversePath.dex,
          isMultiPath: inversePath.isMultiPath,
          pool: inversePath.pool
        })
      );
    }
  }
  emit SetSwapRoute(tokenIn, tokenOut, path, reversePath);
}
```

Recommendation

It is recommended to correct the misspelled names to align with the intended term "router". Consistent and accurate naming conventions enhance code readability, maintainability, and the overall clarity of the contract. Adhering to clear naming practices reduces potential misinterpretation by developers and auditors.

MC - Missing Check

Criticality	Minor / Informative
Location	utils/SwapRouter.sol#L72,160
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

Specifically, a check is missing to ensure that `balancerQueriesAddress` is not `address(0)`

```
balancerQueries = balancerQueriesAddress;
```

Additionally, in `setDexFeeDenominator` a check is missing to ensure that `feeDenominator` is greater than 0. The team could also implement a list of specified denominators to choose from instead of adding it as an input parameter. This will further reduce the risk of adding invalid denominators.

```
function setDexFeeDenominator(uint8 dex, uint256
feeDenominator) external onlyGovernance {
    dexFeeDenominator[dex] = feeDenominator;
    emit SetDexFeeDenominator(dex, feeDenominator);
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

UBUFL - Unintended Balance Used For Liquidity

Criticality	Minor / Informative
Location	utils/LpRouter.sol#L107,306,482
Status	Unresolved

Description

The contract declares `addLiquidity` as a public function. Additionally, `addLiquidity` does not transfer tokens from the user to the contract but instead assumes that they are provided beforehand. This can create situations where users provide tokens in the contract and the tokens are used before they use `addLiquidity`. Since users are able to choose the `amountIn` they could use any amount of tokens stored in the contract. Additionally, excess tokens will be returned to the caller.

```
function addLiquidity(
    address lp,
    address tokenIn,
    uint256 amountIn,
    address recipient,
    DexType dexType
) public nonReentrant returns (uint256 lpAmountOut)
```

Additionally in `_addLiquidityBex` if amountIn is zero or one then `_handleBexTokenSwaps` will return `amount1` calculated by the entire contract's balance.

```
function _addLiquidityBex(
    IBeraPool lp,
    address tokenIn,
    uint256 amountIn,
    address recipient
  ) internal returns (uint256 lpAmountOut) {
    //...
    (amount0, amount1) = _handleBexTokenSwap(tokenIn,
address(lp), tokens, amount0, amount1);
    //...
}

function _handleBexTokenSwap(
    address tokenIn,
    address lp,
    IERC20[] memory tokens,
    uint256 amount0,
    uint256 amount1
) internal returns (uint256 swappedAmount0, uint256
swappedAmount1) {
    if (amount0 > 0 && amount1 > 0) {
        //...
    } else {
        uint256 tokenInBalance =
IERC20(tokenIn).balanceOf(address(this));
        //...
    }
}
```

Recommendation

The team could consider using a strategy that allows users to first approve the tokens to the `LpRouter` and then use `addLiquidity` to utilize only the approved amount.

SAU - Swapped Amount Uninitialized

Criticality	Minor / Informative
Location	utils/LpRouter.sol#L338
Status	Unresolved

Description

`handleTokenSwaps` is used to swap `tokenIn` to a token that can be used to provide liquidity to a specified `lp`. However if either of the initial amounts added is zero or when more than two tokens are added as input the entire contract's balance of `tokenIn` is used to calculate the `swappedAmount1`. The `swappedAmount0` is not calculated, therefore it will always be zero.

```
{
  uint256 tokenInBalance =
IERC20(tokenIn).balanceOf(address(this));
  if (bexLpToTokenIn[lp] != address(0)) {
    token1 = bexLpToTokenIn[lp];
  }
  if (address(tokenIn) != token1) {
    IERC20(tokenIn).safeTransfer(address(swapRouter),
tokenInBalance);
    swappedAmount1 = swapRouter.swapWithDefaultDex(tokenIn,
token1, tokenInBalance, 0, address(this));
  } else {
    swappedAmount1 = tokenInBalance;
  }
  IERC20(token1).forceApprove(beraVault, swappedAmount1);
}
```

Recommendation

The team is advised to consider the situations that can occur when trying to provide liquidity with zero amounts of tokens.

ORA - Overwriting Rewards Amount

Criticality	Minor / Informative
Location	strategies/InfraredStrategy.sol#L108
Status	Unresolved

Description

`harvest` function loops through the `rewardTokensLength` to find a `rewardToken` that is equal to the address of the `asset`. However if multiple `rewardToken` have this address then only the last one will be harvested since `newAssets` is getting overwritten.

```
for (uint256 i = 0; i < rewardTokensLength; i++) {
  address rewardToken = staking.rewardTokens(i);
  uint256 rewardAmount =
IERC20(rewardToken).balanceOf(address(this));
  if (rewardToken == address(asset)) {
    newAssets = rewardAmount;
  } else if (rewardAmount > 0 && rewardToken != wrappedNative)
{
    IERC20(rewardToken).safeTransfer(address(swapRouter),
rewardAmount);
    swapRouter.swapWithDefaultDex(rewardToken, wrappedNative,
rewardAmount, 0, address(this));
  }
}
```

Recommendation

The team could consider adding the amount instead of overwriting it. In that case, the `harvest` function will account for all the possible `asset` tokens instead of the last one.

GRLNS - Gamma Remove Liquidity Not Supported

Criticality	Minor / Informative
Location	utils/LpRouter.sol#L132
Status	Unresolved

Description

`LpRouter` does not support removing liquidity for the `DexType.GAMMA`. Since `LpRouter` allows for adding liquidity to a `GAMMA` `DexType` there should also be functionality for removing it.

```
function removeLiquidity(
    address lp,
    uint256 lpAmount,
    address recipient,
    address tokenOut,
    DexType dexType
) public override nonReentrant returns (uint256 tokenOutAmount)
{
    address router = routers[uint8(dexType)];
    if (dexType == DexType.BEX) {
        return _removeLiquidityBex(IBeraPool(lp), lpAmount,
recipient, tokenOut);
    } else if (dexType == DexType.STEER) {
        return
_removeLiquiditySteer(ISushiMultiPositionLiquidityManager(lp),
lpAmount, recipient, tokenOut);
    } else if (dexType == DexType.KODIAK_V3) {
        return _removeLiquidityKodiak(IKodiakVaultV1(lp),
lpAmount, recipient, router, tokenOut);
    } else {
        revert UnsupportedDexType();
    }
}
```

Recommendation

The team should consider adding support for removing liquidity for `DexType.GAMMA`.

PRE - Potential Reentrance Exploit

Criticality	Minor / Informative
Location	zappers/ZapperBase.sol#L242,271,283
Status	Unresolved

Description

The contract makes an external call to transfer funds to recipients using the payable transfer method. The recipient could be a malicious contract that has an untrusted code in its fallback function that makes a recursive call back to the original contract. The re-entrance exploit could be used by a malicious user to drain the contract's funds or to perform unauthorized actions. This could happen because the original contract does not update the state before sending funds.

Even if `SwapRouter` and `LpRouter` are protected from reentrancies it would be highly beneficial if zappers are also protected from them, especially since in some cases recipient is a parameter added by the user.

```
function _returnAssets(
    address[] memory tokens,
    address recipient
) internal returns (ReturnedAsset[] memory returnedAssets) {
    //...
    (bool success, ) = recipient.call{value: balance}(new
bytes(0));
    if (!success) revert ETHTransferFailed();
    } else {
    //...
}

function _returnAsset(address token, address recipient)
internal returns (ReturnedAsset[] memory returnedAssets) {
    //...
    returnedAssets = _returnAssets(tokens, recipient);
}

function _returnAssetWithAmount(address token, address
recipient, uint256 amount) internal {
    //...
    (bool success, ) = recipient.call{value: amount}(new
bytes(0));
    if (!success) revert ETHTransferFailed();
    //...
}
```

Recommendation

The team is advised to prevent the potential re-entrance exploit as part of the solidity best practices. Some suggestions are:

- Add lockers/mutexes in the method scope. It is important to note that mutexes do not prevent cross-function reentrancy attacks.
- Do Not allow contract addresses to receive funds.
- Proceed with the external call as the last statement of the method, so that the state will have been updated properly during the re-entrance phase.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	strategies/ArberaStrategy.sol#L127
Status	Unresolved

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 _rewardToken
```

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/stable/style-guide.html#naming-conventions.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	zappers/SteerZapper.sol#L30,31,32,33,34,35,36 zappers/KodiakZapper.sol#L30,32,33,34,35,36 zappers/InfraredZapper.sol#L26,27,28,29,30,31,32 zappers/ArberaZapper.sol#L25,26,27,28,29,30,31
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
address wrappedNative
address stablecoin
address swapRouter
address lpRouter
address feeRecipient
uint16 zapInFee
uint16 zapOutFee
address strategist
address governance
address timelock
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	strategies/StrategyFactoryBase.sol#L269 strategies/StrategyBase.sol#L423
Status	Unresolved

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 {
      strategyAddress := create(0, add(bytecode, 0x20),
mload(bytecode))

      if iszero(extcodesize(strategyAddress)) {
        revert(0, 0)
      }
    }
...
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ZapperBase	Implementation	ReentrancyGuard, IZapper		
		Public	✓	-
	setGovernance	External	✓	onlyGovernance
	setSwapRouter	External	✓	onlyGovernance
	setLpRouter	External	✓	onlyGovernance
	setStableCoin	External	✓	onlyGovernance
	setZapInFee	External	✓	onlyGovernance
	setZapOutFee	External	✓	onlyGovernance
	setFeeRecipient	External	✓	onlyGovernance
		External	Payable	-
	_revertAddressZero	Internal		
	_approveTokenIfNeeded	Internal	✓	
	_safeTransferFromTokens	Internal	✓	
	_divideAmountInRatio	Internal		
	_returnAssets	Internal	✓	
	_returnAsset	Internal	✓	
	_returnAssetWithAmount	Internal	✓	
	_transferFee	Internal	✓	

	swapToAssets	Public	✓	-
	swapFromAssets	Public	✓	-
	zapIn	External	Payable	nonReentrant
	zapOut	External	✓	nonReentrant
VaultFactory	Implementation	IVaultFactory		
		Public	✓	-
	_revertAddressZero	Internal		
	setDev	External	✓	onlyDev
	setWhitelistedStrategyFactory	External	✓	onlyDev
	createVault	External	✓	onlyWhitelisted StrategyFactory
Vault	Implementation	ReentrancyGuard, ERC4626, IVault		
		Public	✓	ERC4626 ERC20
	_revertOnlyGovernance	Internal		
	_revertOnlyTimelock	Internal		
	_revertOnlyController	Internal		
	_revertAddressZero	Internal		
	decimals	Public		-
	setMin	External	✓	onlyGovernance
	setDepositFee	External	✓	onlyGovernance
	setWithdrawFee	External	✓	onlyGovernance

	setGovernance	External	✓	onlyGovernance
	setFeeRecipient	External	✓	onlyGovernance
	setTimelock	External	✓	onlyTimelock
	setController	External	✓	onlyTimelock
	available	Public		-
	totalAssets	Public		-
	earn	Public	✓	nonReentrant
	harvest	External	✓	nonReentrant onlyController
	_withdraw	Internal	✓	
	_deposit	Internal	✓	
	deposit	Public	✓	nonReentrant
	redeem	Public	✓	nonReentrant
SwapRouter	Implementation	ReentrancyGuard, ISwapRouter		
		Public	✓	-
	validateSwapParams	Internal		
	validateSwapWithPathParams	Internal		
	setGovernance	Public	✓	onlyGovernance
	setDefaultDex	External	✓	onlyGovernance
	setRouter	External	✓	onlyGovernance
	setFactory	External	✓	onlyGovernance
	setDexFeeDenominator	External	✓	onlyGovernance

	setPool	External	✓	onlyGovernance
	setSwapRoute	External	✓	onlyGovernance
	swapWithDefaultDex	External	✓	resetPathLength
	swap	Public	✓	nonReentrant
	swapWithPathWithDefaultDex	Public	✓	nonReentrant
	swapWithPath	Public	✓	nonReentrant
	getQuoteWithDefaultDex	External	✓	-
	getQuote	Public	✓	-
	getQuoteWithPathWithDefaultDex	External		-
	getQuoteWithPath	Public		-
	_revertAddressZero	Internal		
	_revertZeroAmount	Internal		
	_revertInvalidPathLength	Internal		
	_findMostLiquidV3Pool	Internal		
	_swapWithRoute	Internal	✓	
	_swapBex	Internal	✓	
	_swapV3	Internal	✓	
	_swapV3WithPath	Internal	✓	
	_swapV2	Internal	✓	
	_swapV2WithPath	Internal	✓	
	_getQuoteV3	Internal		
	_getQuoteV3WithPath	Internal		
	_getBexQuote	Internal	✓	
	_getQuoteV2	Internal		

	_getQuoteV2WithPath	Internal		
StrategyFactoryBase	Implementation	IStrategyFactory		
		Public	✓	-
	revertOnlyDev	Private		
	_revertAddressZero	Internal		
	setDev	External	✓	onlyDev
	setVaultFactory	External	✓	onlyDev
	setControllerFactory	External	✓	onlyDev
	_createControllerAndVault	Internal	✓	
	_setupVault	Internal	✓	
	revertIfNonInitializedParams	Private		
	_encodeParamsAndController	Private		
	createVault	External	✓	onlyDev onlyNewAsset
	_deployStrategyByteCode	Internal	✓	
StrategyBase	Implementation	ReentrancyGuard, IStrategy		
		Public	✓	-
		External	Payable	-
	balanceOfAsset	Public		-
	balanceOfPool	Public		-
	balanceOf	Public		-
	_revertAddressZero	Internal		

	_revertOnlyGovernance	Internal		
	_revertOnlyTimelock	Internal		
	_revertOnlyController	Internal		
	_revertOnlyBenevolent	Internal		
	_swapBGTToAsset	Internal	✓	
	whitelistHarvester	External	✓	onlyBenevolent
	revokeHarvester	External	✓	onlyBenevolent
	setWithdrawalDevFundFee	External	✓	onlyTimelock
	setWithdrawalTreasuryFee	External	✓	onlyTimelock
	setPerformanceDevFee	External	✓	onlyTimelock
	setPerformanceTreasuryFee	External	✓	onlyTimelock
	setStrategist	External	✓	onlyGovernance
	setGovernance	External	✓	onlyGovernance
	setTimelock	External	✓	onlyTimelock
	setController	External	✓	onlyTimelock
	setSwapRouter	External	✓	onlyGovernance
	setLpRouter	External	✓	onlyGovernance
	setZapper	External	✓	onlyGovernance
	deposit	Public	✓	-
	getHarvestable	External		-
	harvest	Public	✓	nonReentrant onlyBenevolent
	_withdrawSome	Internal	✓	
	withdraw	External	✓	nonReentrant onlyController

	withdraw	External	✓	nonReentrant onlyController
	withdrawForSwap	External	✓	nonReentrant onlyController
	withdrawAll	External	✓	nonReentrant onlyController
	_withdrawAll	Internal	✓	
	execute	Public	Payable	onlyTimelock
	_distributePerformanceFeesBasedAmountAndDeposit	Internal	✓	
SteerZapper	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	-
	swapFromAssets	Public	✓	-
SteerStrategy	Implementation	StrategyBase		
		Public	✓	StrategyBase
	deposit	Public	✓	-
	balanceOfPool	Public		-
	_withdrawSome	Internal	✓	
SteerFactory	Implementation	StrategyFactoryBase		
		Public	✓	StrategyFactoryBase
	createVaultWithParams	External	✓	onlyDev onlyNewAsset

LpRouter	Implementation	ReentrancyGuard, ILpRouter		
		Public	✓	-
	setGovernance	Public	✓	onlyGovernance
	setSwapRouter	External	✓	onlyGovernance
	setBexLpToTokenIn	External	✓	onlyGovernance
	setRouter	External	✓	onlyGovernance
	addLiquidity	Public	✓	nonReentrant
	removeLiquidity	Public	✓	nonReentrant
	_revertAddressZero	Internal		
	_returnAssets	Internal	✓	
	_swapAndReturnAssets	Internal	✓	
	_swapAndReturnAssets	Internal	✓	
	_approveTokenIfNeeded	Internal	✓	
	_divideAmountInRatio	Internal		
	_getAmountsForLiquidityInRatio	Internal	✓	
	_swapTokensForLiquidity	Internal	✓	
	_isComposableStablePool	Internal		
	_handleBexTokenSwap	Internal	✓	
	_prepareJoinPoolRequest	Internal		
	_prepareExitPoolRequest	Internal		
	_addLiquidityBex	Internal	✓	
	_addLiquidityKodiak	Internal	✓	
	_addLiquiditySteer	Internal	✓	

	_addLiquidityGamma	Internal	✓	
	_removeLiquiditySteer	Internal	✓	
	_removeLiquidityKodiak	Internal	✓	
	_removeLiquidityBex	Internal	✓	
KodiakZapper	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	-
	swapFromAssets	Public	✓	-
KodiakStrategy	Implementation	StrategyBase		
		Public	✓	StrategyBase
	deposit	Public	✓	-
	balanceOfPool	Public		-
	_withdrawSome	Internal	✓	
KodiakFactory	Implementation	StrategyFactoryBase		
		Public	✓	StrategyFactoryBase
	createVaultWithParams	External	✓	onlyDev onlyNewAsset
InfraredZapper	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	-

	_swapToAssetsLp	Internal	✓	
	swapFromAssets	Public	✓	-
	_swapFromAssetsLp	Internal	✓	
	setAssetInfo	External	✓	onlyGovernance
InfraredStrategy	Implementation	StrategyBase		
		Public	✓	StrategyBase
	deposit	Public	✓	nonReentrant
	getHarvestable	External		-
	harvest	Public	✓	onlyBenevolent
	balanceOfPool	Public		-
	_withdrawSome	Internal	✓	
	setRewardTokensLength	External	✓	onlyGovernance
	setStaking	External	✓	onlyGovernance
InfraredFactory	Implementation	StrategyFactoryBase		
		Public	✓	StrategyFactoryBase
	deployStrategyWithParamsAndController	Internal	✓	
	createVaultWithParams	External	✓	onlyDev onlyNewAsset
ControllerFactory	Implementation	IControllerFactory		
		Public	✓	-

	_revertAddressZero	Internal		
	setDev	External	✓	onlyDev
	setWhitelistedStrategyFactory	External	✓	onlyDev
	createController	External	✓	onlyWhitelisted StrategyFactory
Controller	Implementation	ReentrancyGuard, IController		
		Public	✓	-
	_revertAddressZero	Internal		
	_revertOneAddressZero	Internal		
	_revertOnlyGovernance	Internal		
	_revertOnlyStrategist	Internal		
	_revertOnlyTimelock	Internal		
	setDevFund	Public	✓	onlyGovernance
	setTreasury	Public	✓	onlyGovernance
	setStrategist	Public	✓	onlyGovernance
	setGovernance	Public	✓	onlyGovernance
	setTimelock	Public	✓	onlyTimelock
	setVault	Public	✓	onlyStrategist
	approveStrategy	Public	✓	onlyTimelock
	revokeStrategy	Public	✓	onlyGovernance
	setStrategy	Public	✓	nonReentrant onlyStrategist
	earn	Public	✓	nonReentrant

	balanceOf	External		-
	withdrawAll	Public	✓	nonReentrant onlyStrategist
	inCaseTokensGetStuck	Public	✓	onlyGovernance
	inCaseStrategyTokenGetStuck	Public	✓	onlyGovernance
	withdraw	Public	✓	nonReentrant onlyVault
ArberaZapper	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	-
	swapFromAssets	Public	✓	-
	swapToAssetsWithBond	Public	✓	-
	swapFromAssetsWithBond	Public	✓	-
	_getAmounts	Internal		
	zapIn	External	Payable	nonReentrant
	zapInWithBond	External	Payable	nonReentrant
	zapOutWithBond	External	✓	nonReentrant
ArberaStrategy	Implementation	StrategyBase		
		Public	✓	StrategyBase
	deposit	Public	✓	-
	getHarvestable	External		-
	harvest	Public	✓	onlyBenevolent
	balanceOfPool	Public		-

	_withdrawSome	Internal	✓	
	setRewardToken	External	✓	onlyGovernance
ArberaFactory	Implementation	StrategyFactoryBase		
		Public	✓	StrategyFactoryBase
	createVaultWithParams	External	✓	onlyDev onlyNewAsset

Inheritance Graph

For the detailed graph file, please refer to the following link:

inheritance_graph.png

Summary

The BeraTrax protocol implements a modular and secure yield-generation mechanism through its core components, Vaults, Controllers, Strategies, Factories, and Zappers. This audit investigates security vulnerabilities, evaluates business logic consistency, and identifies potential improvements to enhance system robustness and operational efficiency.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

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

cyberscope.io