



Cyberscope

Audit Report

BeraTrax

January 2025

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

Audited by © cyberscope

Table of Contents

Table of Contents	1
Risk Classification	4
Review	5
Audit Updates	5
Source Files	5
Overview	7
Controller	7
Factories	7
Strategies	7
Vault	8
Zappers	8
Findings Breakdown	9
Diagnostics	10
SRO - Swap Route Overwrite	12
Description	12
Recommendation	13
PFLRA - Potential Flash Loan Reentrancy Attacks	14
Description	14
Recommendation	15
USSP - User Specified Swap Parameters	16
Description	16
Recommendation	18
CCR - Contract Centralization Risk	19
Description	19
Recommendation	20
ETT - Excess Token Transfer	22
Description	22
Recommendation	23
IDI - Immutable Declaration Improvement	24
Description	24
Recommendation	24
MBV - Missing Balance Validation	25
Description	25
Recommendation	25
MSV - Missing Strategy Validation	26
Description	26
Recommendation	26
MVN - Misspelled Variable Naming	27
Description	27

Recommendation	28
MU - Modifiers Usage	29
Description	29
Recommendation	29
PBV - Percentage Boundaries Validation	30
Description	30
Recommendation	30
PLPI - Potential Liquidity Provision Inadequacy	31
Description	31
Recommendation	33
UTPD - Unverified Third Party Dependencies	35
Description	35
Recommendation	35
ZD - Zero Division	36
Description	36
Recommendation	36
L02 - State Variables could be Declared Constant	37
Description	37
Recommendation	37
L05 - Unused State Variable	38
Description	38
Recommendation	38
L09 - Dead Code Elimination	39
Description	39
Recommendation	39
L14 - Uninitialized Variables in Local Scope	40
Description	40
Recommendation	40
L15 - Local Scope Variable Shadowing	41
Description	41
Recommendation	41
L16 - Validate Variable Setters	42
Description	42
Recommendation	42
L17 - Usage of Solidity Assembly	43
Description	43
Recommendation	43
L19 - Stable Compiler Version	44
Description	44
Recommendation	45
Functions Analysis	46
Inheritance Graph	66

Flow Graph	67
Summary	68
Disclaimer	69
About Cyberscope	70

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

Source Files

Filename	SHA256
zappers/ZapperBase.sol	cd1af4dbaf1104031a8e879ac0cece6ecd4 2e2c35e5f5e8fbbef627e684caf3f
zappers/SteerZapper.sol	318afa5dc453843f77a59557509a089d731 73dac188dccd25b1e838af6a04c65
zappers/KodiakZapper.sol	696804144d3fe9bed4665497e8b6b2cd05 1f2d09e0d2bf81bc5d62e29ffdbc54
zappers/InfraredZapper.sol	0a3f5965e23fe870dde2384f46fa598198d b66ee9ae4aac0f08728a46d58c664
zappers/ArberaZapper.sol	09fe633b94fedcccf357fd18e5d3ee6cc18e e12400e0a54f186ea4f1c65c0e92
vaults/Vault.sol	e07487342d4878fb929d49906214a91776 b280b5609a003ecef86ab43e10df
utils/SwapRouter.sol	2d4616475e31128bc2579ccf967a06bd0d 39127e62e3cfa6046980fe7f39eaa5
utils/LpRouter.sol	4293952c4f84ff0867e877467ab353e34c0 5c1b27ebffcbdfb57f85bcd68130d
strategies/StrategyBase.sol	4b94694243a5d3137d79ee3b6cdb42475 93451ede3e9393ab0083b1ed5288b2e

strategies/SteerStrategy.sol	1db238de574c249a12e6358d1c5b5b7e0 59d4db64861d649b3aed5c1eb41066e
strategies/KodiakStrategy.sol	55b4be81fa9ebfad479f1b9b7216398bc90 934b7736538da35484f2b9c326cc1
strategies/InfraredStrategy.sol	05d4617d465c8d8b9b6653949522acdae a15c4d5ad22d052abc58725a03d7e78
strategies/ArberaStrategy.sol	3e77e3167c3dd6f5013254ce3866905b3e 5594bcd843dc53ae40544219a273fa
factories/VaultAndControllerFactory.sol	307d23fb5c5ff0b6d0c10f60faa9d10dbe36 f075b0682c6ce8c5f51f695b42e9
factories/StrategyFactoryBase.sol	f9141c51fff6d6d3932ca57fcf393050e3279 6524b1e0a1f74f6f5387d00ba22
factories/SteerFactory.sol	fda2f286f4c0bd693fa950649faf5363898c 2edd9f4d670344fe2bf836a76643
factories/KodiakFactory.sol	a55c5b6e43672461f5acbae7b494b4dcd2 71826644d7bb0f6bd4d309601305c0
factories/InfraredFactory.sol	224d9578b6912fdf92bb35e9a7a6a98d6b 3178cf653f72d7ebb41fb351bf4934
factories/ArberaFactory.sol	fdf03a3bf820aa0ac052bf3b63a8d9e3d36 e26f27eee20742bd555b2bf2c9e55
controllers/Controller.sol	03614e425e2c40c3370fbb62699733e94e 8e4b836ef185d3dfde6535466501b2

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.

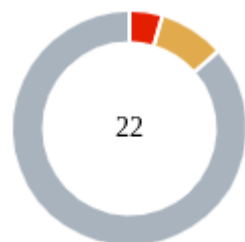
Vault

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	1
Medium	2
Minor / Informative	19

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	2	0	0	0
Minor / Informative	18	1	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	SRO	Swap Route Overwrite	Unresolved
●	PFLRA	Potential Flash Loan Reentrancy Attacks	Unresolved
●	USSP	User Specified Swap Parameters	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	ETT	Excess Token Transfer	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MBV	Missing Balance Validation	Unresolved
●	MSV	Missing Strategy Validation	Acknowledged
●	MVN	Misspelled Variable Naming	Unresolved
●	MU	Modifiers Usage	Unresolved
●	PBV	Percentage Boundaries Validation	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	UTPD	Unverified Third Party Dependencies	Unresolved
●	ZD	Zero Division	Unresolved

●	L02	State Variables could be Declared Constant	Unresolved
●	L05	Unused State Variable	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

SRO - Swap Route Overwrite

Criticality	Critical
Location	utils/SwapRouter.sol#L14
Status	Unresolved

Description

The contract is at risk of losing data when setting swap routes for token pairs in the `setSwapRoute` function. This happens because the function reuses the same storage reference (`swapRoute`) for both the forward and reverse paths. Specifically, when the reverse path is being assigned to `swapRoutes[tokenOut][tokenIn]`, the storage pointer originally referencing `swapRoutes[tokenIn][tokenOut]` is overwritten. As a result, any data in the forward route can be unintentionally discarded or overwritten, leading to unexpected behaviour or inconsistencies in the swap route logic.

This issue can cause the contract to fail to properly maintain distinct forward and reverse routes, which may disrupt token swaps or lead to errors during operations involving these routes.

```
function setSwapRoute (
  address tokenIn,
  address tokenOut,
  SwapRoutePath[] memory path,
  bool reversePath
) external onlyGovernance {
  SwapRoutePath[] storage swapRoute = swapRoutes[tokenIn][tokenOut];
  if (swapRoute.length > 0) delete swapRoutes[tokenIn][tokenOut];
  for (uint256 i = 0; i < path.length; i++) {
    swapRoute.push(path[i]);
  }

  if (reversePath) {
    swapRoute = swapRoutes[tokenOut][tokenIn];
    for (uint256 i = 0; i < path.length; i++) {
      SwapRoutePath memory inversePath = path[path.length - i - 1];
      swapRoute.push(
        SwapRoutePath({
          tokenIn: inversePath.tokenOut,
          tokenOut: inversePath.tokenIn,
          dex: inversePath.dex,
          isMultiPath: inversePath.isMultiPath
        })
      );
    }
  }
  emit SetSwapRoute(tokenIn, tokenOut, path, reversePath);
}
```

Recommendation

It is recommended to use separate, distinct storage references for the forward (`swapRoutes[tokenIn][tokenOut]`) and reverse (`swapRoutes[tokenOut][tokenIn]`) paths. Avoid reassigning the same reference (`swapRoute`) for both paths. Managing these routes independently ensures that the forward path remains unaffected when setting the reverse path. This approach preserves data integrity, reduces confusion, and ensures consistent routing behaviour.

PFLRA - Potential Flash Loan Reentrancy Attacks

Criticality	Medium
Location	vaults/Vault.sol#L203 utils/SwapRouter.sol#L303 controllers/Controller.sol#L192
Status	Unresolved

Description

The contract is vulnerable to potential arbitrage and reentrancy attacks due to the absence of the `nonReentrant` modifiers and the frequent use of external calls across crucial functions. For instance, functions such as `earn`, `withdraw`, and `deposit` involve transferring assets and interacting with external contracts through the swap flows. This creates opportunities for attackers to exploit the system through reentrant calls. By manipulating the state of the Vault, Controller, or Strategy between external calls, an attacker could extract more assets than intended, manipulate exchange rates for arbitrage opportunities, or drain funds entirely.

Specifically, a malicious user could leverage flash loans to perform reentrant calls, arbitrarily modifying the reserves of the Vault between function executions. By exploiting this mechanism, an attacker could manipulate the state of the contracts, allowing them to withdraw a larger amount of assets than initially deposited. The presence of these vulnerabilities, combined with the possibility of flash loans, amplifies the risk of sophisticated attacks that could severely impact the protocol's functionality and user funds.

```
function deposit(uint256 assets, address receiver, uint256 minShares)
public returns (uint256 shares) {
    shares = super.deposit(assets, receiver);
    if (shares < minShares) revert InsufficientOutputShares(shares,
minShares);
}

function redeem(uint256 shares, address receiver, address owner,
uint256 minAssets) public returns (uint256 assets) {
    assets = super.redeem(shares, receiver, owner);
    if (assets < minAssets) revert InsufficientOutputAssets(assets,
minAssets);
}
...

function earn(address asset, uint256 amount) public {
    if (asset == address(0)) revert ZeroAddress();
    if (amount > 0) {
        address strategy = strategies[asset];
        IERC20(asset).safeTransfer(strategy, amount);
        IStrategy(strategy).deposit();
        emit Earned(asset, amount);
    }
    ...
}

function swapWithPath(
    address[] calldata path,
    uint256 amountIn,
    uint256 amountOutMinimum,
    address recipient,
    DexType dex
) public returns (uint256 amountOut) {
    ...
}
```

Recommendation

It is recommended to include the `nonReentrant` modifier on all critical functions that involve external calls or state modifications, such as `earn`, `withdraw`, and `deposit`. This will prevent reentrant calls that could disrupt the contract's state.

Additionally, the contract should validate its state before and after external interactions to ensure consistency and mitigate the risk of manipulation. Slippage controls or withdrawal limits could be implemented to safeguard against exchange rate manipulation, especially during volatile conditions. By minimising the number of external calls in sensitive workflows and protect the contract against reentrancy and flash loan scenarios, the risk of exploitation can be significantly reduced, ensuring the safety and reliability of the system.

USSP - User Specified Swap Parameters

Criticality	Medium
Location	utils/SwapRouter.sol#L utils/SwapRouter.sol#L185,204,286,303
Status	Unresolved

Description

The contract is designed to allow users to execute swaps by specifying critical parameters such as the swap path, token addresses, and router addresses. While this flexibility is beneficial, it introduces risks where malicious users could exploit these parameters. For example, they might provide invalid paths, unsupported token addresses, or malicious router addresses, potentially causing the contract to behave unexpectedly or fail. This could result in the manipulation of contract state, disruption of swap logic, or loss of funds.

```
function swapWithDefaultDex(  
    address tokenIn,  
    address tokenOut,  
    uint256 amountIn,  
    uint256 amountOutMinimum,  
    address recipient  
) external resetPathLength returns (uint256 amountOut) {  
    return  
        _swapWithRoute(tokenIn, tokenOut, amountIn, amountOutMinimum,  
            recipient);  
}
```

```
function swap(
    address tokenIn,
    address tokenOut,
    uint256 amountIn,
    uint256 amountOutMinimum,
    address recipient,
    DexType dex
) public returns (uint256 amountOut) {
    if (
        tokenIn == address(0) || tokenOut == address(0) || recipient ==
address(0)
    ) revert ZeroAddress();
    if (amountIn == 0) revert ZeroAmount();

    address router = routers[uint8(dex)];
    if (router == address(0)) revert RouterNotSupported();

    return _swapV3(tokenIn, tokenOut, amountIn, amountOutMinimum,
recipient, router, factories[uint8(dex)]);
}
```

```
function swapWithPathWithDefaultDex(
    address[] calldata path,
    uint256 amountIn,
    uint256 amountOutMinimum,
    address recipient
) public returns (uint256 amountOut) {
    return
        swapWithPath(path, amountIn, amountOutMinimum, recipient,
defaultDex);
}
```

```
function swapWithPath(  
    address[] calldata path,  
    uint256 amountIn,  
    uint256 amountOutMinimum,  
    address recipient,  
    DexType dex  
) public returns (uint256 amountOut) {  
    if (path.length < 2) revert InvalidPathLength();  
    if (amountIn == 0) revert ZeroAmount();  
    if (recipient == address(0)) revert ZeroAddress();  
  
    return  
        _swapV3WithPath(  
            path,  
            amountIn,  
            amountOutMinimum,  
            recipient,  
            routers[uint8(dex)],  
            factories[uint8(dex)]  
        );  
}
```

Recommendation

It is recommended to implement robust validation mechanisms for all user-specified parameters. Specifically, ensure that paths contain valid token addresses and supported routes, router addresses are verified against a whitelist of trusted routers, and token interactions adhere to expected behaviours. These safeguards will minimise the risk of exploitation while maintaining the intended functionality of the contract.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	controllers/Controller.sol#L120,177 vaults/Vault.sol#L102
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 setGovernance(address governanceAddress) external
onlyGovernance {
    if (governanceAddress == address(0)) revert ZeroAddress();
    address old = governance;
    governance = governanceAddress;
    emit GovernanceChanged(old, governanceAddress);
}

function setTimelock(address timelockAddress) external onlyTimelock {
    if (timelockAddress == address(0)) revert ZeroAddress();
    address old = timelock;
    timelock = timelockAddress;
    emit TimelockChanged(old, timelockAddress);
}

function setController(address controllerAddress) external
onlyTimelock {
    if (controllerAddress == address(0)) revert ZeroAddress();
    address old = controller;
    controller = controllerAddress;
    emit ControllerChanged(old, controllerAddress);
}

function setStrategist(address strategistAddress) public
onlyGovernance {
    if (strategistAddress == address(0)) revert ZeroAddress();
    address old = strategist;
    strategist = strategistAddress;
    emit StrategistChanged(old, strategistAddress);
}

function setStrategy(address asset, address strategy) public
onlyStrategist {
    if (asset == address(0) || strategy == address(0)) revert
ZeroAddress();
    if (!approvedStrategies[asset][strategy]) revert
StrategyNotApproved();

    address current = strategies[asset];
    strategies[asset] = strategy;
    if (current != address(0)) {
        IStrategy(current).withdrawAll();
    }
    emit StrategySet(asset, strategy);
}
```

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.

ETT - Excess Token Transfer

Criticality	Minor / Informative
Location	utils/SwapRouter.sol#L570
Status	Unresolved

Description

The contract performs swaps and transfers the entire balance of the swapped tokens to the recipient without verifying if this balance corresponds exactly to the intended output amount of the swap. Even if the contract is not designed to hold or keep funds, this behaviour can lead to scenarios where more tokens than intended are transferred. This could result in unintended behaviour, inefficiencies, or a loss of tokens that were not meant to be swapped or sent.

```
for (uint256 i = 0; i < route.length; i++) {
    SwapRoutePath memory path = route[i];
    if (path.isMultiPath) {
        // recursive call, be careful here
        amountOut = _swapWithRoute(
            path.tokenIn,
            path.tokenOut,
            amountIn,
            0,
            address(this)
        );
    } else {
        amountOut = swap(
            path.tokenIn,
            path.tokenOut,
            amountIn,
            0,
            address(this),
            path.dex
        );
    }
    amountIn = amountOut;
}
amountOut = IERC20(tokenOut).balanceOf(address(this));
if (amountOut < amountOutMinimum)
    revert InsufficientOutputAmount(amountOut, amountOutMinimum);
IERC20(tokenOut).safeTransfer(recipient, amountOut);
return amountOut;
```

Recommendation

It is recommended to transfer only the actual swapped amount to the recipient. This can be achieved by tracking the difference in token balances before and after the swap operation to ensure that only the exact output amount is transferred. Implementing this safeguard will enhance the contract's accuracy, prevent over-transfers, and maintain system reliability.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	StrategyBase.sol#L95 strategies/StrategyBase.sol#L95
Status	Unresolved

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.

```
wrappedNative
```

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.

MBV - Missing Balance Validation

Criticality	Minor / Informative
Location	vaults/Vault.sol#L150
Status	Unresolved

Description

The contract is designed to transfer available funds to a controller for further handling via the `earn` function. However, the function does not include a mechanism to revert or provide an error message in cases where the available balance (`_bal`) is zero. This can result in silent execution without any actionable feedback to the caller, which may cause confusion or hinder debugging efforts. Additionally, failing to notify the caller when no funds are available for transfer could lead to operational inefficiencies or the assumption that the function executed as expected.

```
function earn() public {
    uint256 _bal = available();
    if (_bal > 0) {
        IERC20(asset()).safeTransfer(controller, _bal);
        IController(controller).earn(asset(), _bal);
    }
}
```

Recommendation

It is recommended to implement a check that explicitly reverts the transaction or emits an error message when the available balance is zero. This ensures that the function provides clear feedback to the caller and avoids silent failures. Adding this validation improves transparency, enhances the user experience, and reduces potential confusion during operation.

MSV - Missing Strategy Validation

Criticality	Minor / Informative
Location	controllers/Controller.sol#L192
Status	Acknowledged

Description

The contract is designed to facilitate the redirection of funds to investment strategies via the `earn` function, which takes `asset` and `amount` as parameters. This function locates the corresponding strategy for the given `asset` but fails to validate whether the retrieved `strategy` address actually exists or is valid. Without this check, if a strategy for the specified `asset` 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 asset, uint256 amount) public {
    if (asset == address(0)) revert ZeroAddress();
    if (amount > 0) {
        address strategy = strategies[asset];
        IERC20(asset).safeTransfer(strategy, amount);
        IStrategy(strategy).deposit();
        emit Earned(asset, 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.

MVN - Misspelled Variable Naming

Criticality	Minor / Informative
Location	utils/SwapRouter.sol#L149
Status	Unresolved

Description

The contract is designed to manage swap routes through the `setSwapRoute` function. However, within this function, the word "rout" is used instead of "route" in the variable name `swapRoute`. 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 swapRoute = swapRoutes[tokenIn][tokenOut];  
    if (swapRoute.length > 0) delete swapRoutes[tokenIn][tokenOut];  
    for (uint256 i = 0; i < path.length; i++) {  
        swapRoute.push(path[i]);  
    }  
  
    if (reversePath) {  
        swapRoute = swapRoutes[tokenOut][tokenIn];  
        for (uint256 i = 0; i < path.length; i++) {  
            SwapRoutePath memory inversePath = path[path.length - i - 1];  
            swapRoute.push(  
                SwapRoutePath({  
                    tokenIn: inversePath.tokenOut,  
                    tokenOut: inversePath.tokenIn,  
                    dex: inversePath.dex,  
                    isMultiPath: inversePath.isMultiPath  
                })  
            );  
        }  
    }  
    emit SetSwapRoute(tokenIn, tokenOut, path, reversePath);  
}
```

Recommendation

It is recommended to correct the misspelled variable name `swapRoute` to align with the intended term "route." 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.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	vaults/Vault.sol#L125 utils/SwapRouter.sol#L310 controllers/Controller.sol#L112
Status	Unresolved

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.

```
if (controllerAddress == address(0)) revert ZeroAddress();  
...  
if (path.length < 2) revert InvalidPathLength();  
if (amountIn == 0) revert ZeroAmount();  
if (recipient == address(0)) revert ZeroAddress();  
...  
if (treasuryAddress == address(0)) revert ZeroAddress();  
...
```

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.

PBV - Percentage Boundaries Validation

Criticality	Minor / Informative
Location	strategies/StrategyBase.sol#L185
Status	Unresolved

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 setWithdrawalDevFundFee(uint16 fee) external onlyTimelock {
    uint16 old = withdrawalDevFundFee;
    withdrawalDevFundFee = fee;
    emit WithdrawalDevFundFeeChanged(old, fee);
}
```

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	utils/SwapRouter.sol#L532
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) {  
    currentPathLength++;  
    ...  
}  
  
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(  
    ...  
)  
  
function _swapV2WithPath(  
    address[] memory path,  
    uint256 amountIn,  
    uint256 amountOutMinimum,  
    address recipient,  
    address router  
) internal returns (uint256 amountOut) {  
    ...  
}  
  
function _swapCamelotV3(  
    ...  
)
```

```
    address tokenIn,  
    address tokenOut,  
    uint256 amountIn,  
    uint256 amountOutMinimum,  
    address recipient,  
    address router  
    ) internal returns (uint256 amountOut) {  
        ...  
    }  
  
function _swapCroc(  
    address tokenIn,  
    address tokenOut,  
    uint256 amountIn,  
    uint256 amountOutMinimum,  
    address recipient,  
    address router,  
    address crocQuery  
    ) internal returns (uint256 amountOut) {  
        ...  
    }  
  
function _swapCrocWithPath(  
    address[] memory path,  
    uint256 amountIn,  
    uint256 amountOutMinimum,  
    address recipient,  
    address router,  
    address crocQuery  
    ) 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.

UTPD - Unverified Third Party Dependencies

Criticality	Minor / Informative
Location	strategies/StrategyBase.sol#L151 strategies/InfraredStrategy.sol#L79
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 {  
    ...  
    staking.stake(balance);  
}
```

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.

ZD - Zero Division

Criticality	Minor / Informative
Location	zappers/ZapperBase.sol#L156
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

```
function _divideAmountInRatio(  
    uint256 amount,  
    uint256 ratio0,  
    uint256 ratio1  
) internal pure returns (uint256, uint256) {  
    uint256 totalRatio = ratio0 + ratio1;  
    uint256 amount0 = (amount * ratio0) / totalRatio;  
    ...  
}
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	StrategyBase.sol#L38,40 strategies/StrategyBase.sol#L38,40
Status	Unresolved

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.

```
uint16 public withdrawalDevFundMax = 1000  
uint16 public withdrawalTreasuryMax = 1000
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	KodiakZapper.sol#L31
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
uint256[4] minInAmounts = [uint256(0), uint256(0), uint256(0),  
uint256(0)]
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	ZapperBase.sol#L133,156,172 strategies/InfraredStrategy.sol#L126 InfraredStrategy.sol#L126
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _approveTokenIfNeeded(address tokenAddress, address
spenderAddress) internal {
    if (IERC20(tokenAddress).allowance(address(this), spenderAddress) ==
0) {
        IERC20(tokenAddress).approve(spenderAddress, type(uint256).max);
    }
}

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	ZapperBase.sol#L176
Status	Unresolved

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	strategies/SteerStrategy.sol#L31,33,34 strategies/InfraredStrategy.sol#L47,49,50 strategies/ArberaStrategy.sol#L41,43,44 SteerZapper.sol#L32,33,34,35,36 SteerStrategy.sol#L31,33,34 KodiakZapper.sol#L35,37,38 interfaces/IController.sol#L103,106,109 InfraredZapper.sol#L21,22,23,24 InfraredStrategy.sol#L47,49,50 ArberaZapper.sol#L25,26,27,28 ArberaStrategy.sol#L41,43,44 ArberaFactory.sol#L39
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 swapRouter
address lpRouter
address dev
address stablecoin
address strategist
address governance
address timelock
address vaultAndControllerFactory
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	vaults/Vault.sol#L54,55,56 VaultAndControllerFactory.sol#L28 Vault.sol#L54,55,56 SwapRouter.sol#L59,60
Status	Unresolved

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.

```
governance = governanceAddress
timelock = timelockAddress
controller = controllerAddress
dev = devAddress
wrappedNative = wrappedNativeAddress
```

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	StrategyFactoryBase.sol#L226 StrategyBase.sol#L348 strategies/StrategyBase.sol#L348
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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	ZapperBase.sol#L2 vaults/Vault.sol#L2 VaultAndControllerFactory.sol#L2 Vault.sol#L2 SwapRouter.sol#L2 StrategyFactoryBase.sol#L2 StrategyBase.sol#L2 strategies/StrategyBase.sol#L2 strategies/SteerStrategy.sol#L2 strategies/KodiakStrategy.sol#L2 strategies/InfraredStrategy.sol#L2 strategies/ArberaStrategy.sol#L2 SteerZapper.sol#L2 SteerStrategy.sol#L2 SteerFactory.sol#L2 LpRouter.sol#L2 KodiakZapper.sol#L2 KodiakStrategy.sol#L2 KodiakFactory.sol#L2 interfaces/IVaultAndControllerFactory.sol#L2 interfaces/IVault.sol#L2 interfaces/ISwapRouter.sol#L2 interfaces/ILpRouter.sol#L2 interfaces/IController.sol#L2 InfraredZapper.sol#L2 InfraredStrategy.sol#L2 controllers/Controller.sol#L2 Controller.sol#L2 ArberaZapper.sol#L2 ArberaStrategy.sol#L2 ArberaFactory.sol#L2
Status	Unresolved

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 ^0.8.20;
```

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
ZapperBase	Implementation	IZapper		
		Public	✓	-
	setDev	External	✓	onlyDev
	setSwapRouter	External	✓	onlyDev
	setLpRouter	External	✓	onlyDev
	setStableCoin	External	✓	onlyDev
		External	Payable	-
	_approveTokenIfNeeded	Internal	✓	
	_safeTransferFromTokens	Internal	✓	
	_divideAmountInRatio	Internal		
	_returnAssets	Internal	✓	
	_returnAsset	Internal	✓	
	swapToAssets	Public	✓	-
	swapFromAssets	Public	✓	-
	zapIn	External	Payable	-
	zapOut	External	✓	-
VaultAndControllerFactory	Implementation	IVaultAndControllerFactory		

		Public	✓	-
	setDev	External	✓	onlyDev
	setStrategyFactory	External	✓	onlyDev
	createVaultAndController	External	✓	onlyStrategyFactory
Vault	Implementation	ERC4626, IVault		
		Public	✓	ERC4626 ERC20
	decimals	Public		-
	setMin	External	✓	onlyGovernance
	setGovernance	External	✓	onlyGovernance
	setTimelock	External	✓	onlyTimelock
	setController	External	✓	onlyTimelock
	available	Public		-
	totalAssets	Public		-
	earn	Public	✓	-
	harvest	External	✓	onlyController
	_withdraw	Internal	✓	
	deposit	Public	✓	-
	redeem	Public	✓	-
SwapRouter	Implementation	ISwapRouter		
		Public	✓	-

	setGovernance	Public	✓	onlyGovernance
	setDefaultDex	External	✓	onlyGovernance
	setRouter	External	✓	onlyGovernance
	setFactory	External	✓	onlyGovernance
	setCrocQuery	External	✓	onlyGovernance
	setSwapRoute	External	✓	onlyGovernance
	swapWithDefaultDex	External	✓	resetPathLength
	swap	Public	✓	-
	swapWithPathWithDefaultDex	Public	✓	-
	swapWithPath	Public	✓	-
	getQuoteWithDefaultDex	External		-
	getQuote	Public		-
	getQuoteWithPathWithDefaultDex	External		-
	getQuoteWithPath	Public		-
	_findMostLiquidV3Pool	Internal		
	_findMostLiquidCrocPool	Internal		
	_swapWithRoute	Internal	✓	
	_swapV3	Internal	✓	
	_swapV3WithPath	Internal	✓	
	_swapV2	Internal	✓	
	_swapV2WithPath	Internal	✓	
	_swapCamelotV3	Internal	✓	

	_swapCroc	Internal	✓	
	_swapCrocWithPath	Internal	✓	
	_getQuoteCroc	Internal		
	_getQuoteCrocWithPath	Internal		
	_getQuoteV3	Internal		
	_getQuoteV3WithPath	Internal		
	_getQuoteV2	Internal		
	_getQuoteV2WithPath	Internal		
StrategyFactoryBase	Implementation	IStrategyFactory		
		Public	✓	-
	setDev	External	✓	onlyDev
	setVaultAndControllerFactory	External	✓	onlyDev
	_createControllerAndVault	Internal	✓	
	_setupVault	Internal	✓	
	createVault	External	✓	onlyDev onlyNewAsset
	_deployStrategyByteCode	Internal	✓	
	setVaultData	External	✓	onlyDev
StrategyBase	Implementation	IStrategy		
		Public	✓	-
	balanceOfAsset	Public		-
	balanceOfPool	Public		-

	balanceOf	Public		-
	_swapBGTToAsset	Internal	✓	
	whitelistHarvester	External	✓	-
	revokeHarvester	External	✓	-
	setWithdrawalDevFundFee	External	✓	onlyTimelock
	setWithdrawalTreasuryFee	External	✓	onlyTimelock
	setPerformanceDevFee	External	✓	onlyTimelock
	setPerformanceTreasuryFee	External	✓	onlyTimelock
	setStrategist	External	✓	onlyGovernance
	setGovernance	External	✓	onlyGovernance
	setTimelock	External	✓	onlyTimelock
	setController	External	✓	onlyTimelock
	deposit	Public	✓	-
	harvest	Public	✓	-
	_withdrawSome	Internal	✓	
	withdraw	External	✓	onlyController
	withdraw	External	✓	onlyController
	withdrawForSwap	External	✓	onlyController
	withdrawAll	External	✓	onlyController
	_withdrawAll	Internal	✓	
	execute	Public	Payable	onlyTimelock
	_distributePerformanceFeesBasedAmountAndDeposit	Internal	✓	

SteerZapper	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	-
	swapFromAssets	Public	✓	-
	_getAmountsForLiquidity	Internal		
SteerStrategy	Implementation	StrategyBase		
		Public	✓	StrategyBase
	deposit	Public	✓	-
	getHarvestable	External		-
	balanceOfPool	Public		-
	_withdrawSome	Internal	✓	
SteerFactory	Implementation	StrategyFactoryBase		
		Public	✓	StrategyFactoryBase
	createVaultWithParams	External	✓	onlyDev onlyNewAsset
LpRouter	Implementation	ILpRouter		
		Public	✓	-
	setGovernance	Public	✓	onlyGovernance
	setDefaultDex	External	✓	onlyGovernance
	setRouter	External	✓	onlyGovernance

	addLiquidityWithDefaultDex	External	✓	-
	addLiquidity	Public	✓	-
	removeLiquidityWithDefaultDex	External	✓	-
	removeLiquidity	Public	✓	-
	_addLiquidityCrocSwap	Internal	✓	
	_removeLiquidityCrocSwap	Internal	✓	
KodiakZapper	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	-
	swapFromAssets	Public	✓	-
	_getTokenBalances	Internal		
	_getKodiakVaultMintAmount	Internal		
	_getAmountsForLiquidity	Internal		
	_addLiquidity	Internal	✓	
KodiakStrategy	Implementation	StrategyBase		
		Public	✓	StrategyBase
	deposit	Public	✓	-
	getHarvestable	External		-
	balanceOfPool	Public		-
	_withdrawSome	Internal	✓	

KodiakFactory	Implementation	StrategyFactoryBase		
		Public	✓	StrategyFactoryBase
	createVaultWithParams	External	✓	onlyDev onlyNewAsset
InfraredZapper	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	-
	swapFromAssets	Public	✓	-
InfraredStrategy	Implementation	StrategyBase		
		Public	✓	StrategyBase
	deposit	Public	✓	-
	getHarvestable	External		-
	harvest	Public	✓	-
	_swapAndAddLiquidity	Internal	✓	
	balanceOfPool	Public		-
	_withdrawSome	Internal	✓	
	setRewardTokensLength	External	✓	onlyGovernance
Controller	Implementation	IController		
		Public	✓	-
	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	✓	onlyStrategist
	earn	Public	✓	-
	balanceOf	External		-
	withdrawAll	Public	✓	onlyStrategist
	inCaseTokensGetStuck	Public	✓	onlyGovernance
	inCaseStrategyTokenGetStuck	Public	✓	onlyGovernance
	withdraw	Public	✓	onlyVault
ArberaZapper	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	-
	swapFromAssets	Public	✓	-
	swapToAssetsWithBond	Public	✓	-
	swapFromAssetsWithBond	Public	✓	-
	_getAmounts	Internal		

	zapInWithBond	External	Payable	-
	zapOutWithBond	External	✓	-
ArberaStrategy	Implementation	StrategyBase		
		Public	✓	StrategyBase
	deposit	Public	✓	-
	getHarvestable	External		-
	balanceOfPool	Public		-
	_withdrawSome	Internal	✓	
ArberaFactory	Implementation	StrategyFactoryBase		
		Public	✓	StrategyFactoryBase
	createVaultWithParams	External	✓	onlyDev onlyNewAsset
ZapperBase	Implementation	IZapper		
		Public	✓	-
	setDev	External	✓	onlyDev
	setSwapRouter	External	✓	onlyDev
	setLpRouter	External	✓	onlyDev
	setStableCoin	External	✓	onlyDev
		External	Payable	-
	_approveTokenIfNeeded	Internal	✓	

	_safeTransferFromTokens	Internal	✓	
	_divideAmountInRatio	Internal		
	_returnAssets	Internal	✓	
	_returnAsset	Internal	✓	
	swapToAssets	Public	✓	-
	swapFromAssets	Public	✓	-
	zapIn	External	Payable	-
	zapOut	External	✓	-
SteerZapper	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	-
	swapFromAssets	Public	✓	-
	_getAmountsForLiquidity	Internal		
KodiakZapper	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	-
	swapFromAssets	Public	✓	-
	_getTokenBalances	Internal		
	_getKodiakVaultMintAmount	Internal		
	_getAmountsForLiquidity	Internal		
	_addLiquidity	Internal	✓	

InfraredZapper	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	-
	swapFromAssets	Public	✓	-
ArberaZapper	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	-
	swapFromAssets	Public	✓	-
	swapToAssetsWithBond	Public	✓	-
	swapFromAssetsWithBond	Public	✓	-
	_getAmounts	Internal		
	zapInWithBond	External	Payable	-
	zapOutWithBond	External	✓	-
Vault	Implementation	ERC4626, IVault		
		Public	✓	ERC4626 ERC20
	decimals	Public		-
	setMin	External	✓	onlyGovernance
	setGovernance	External	✓	onlyGovernance
	setTimelock	External	✓	onlyTimelock

	setController	External	✓	onlyTimelock
	available	Public		-
	totalAssets	Public		-
	earn	Public	✓	-
	harvest	External	✓	onlyController
	_withdraw	Internal	✓	
	deposit	Public	✓	-
	redeem	Public	✓	-
SwapRouter	Implementation	ISwapRouter		
		Public	✓	-
	setGovernance	Public	✓	onlyGovernance
	setDefaultDex	External	✓	onlyGovernance
	setRouter	External	✓	onlyGovernance
	setFactory	External	✓	onlyGovernance
	setCrocQuery	External	✓	onlyGovernance
	setSwapRoute	External	✓	onlyGovernance
	swapWithDefaultDex	External	✓	resetPathLength
	swap	Public	✓	-
	swapWithPathWithDefaultDex	Public	✓	-
	swapWithPath	Public	✓	-
	getQuoteWithDefaultDex	External		-

	getQuote	Public		-
	getQuoteWithPathWithDefaultDex	External		-
	getQuoteWithPath	Public		-
	_findMostLiquidV3Pool	Internal		
	_findMostLiquidCrocPool	Internal		
	_swapWithRoute	Internal	✓	
	_swapV3	Internal	✓	
	_swapV3WithPath	Internal	✓	
	_swapV2	Internal	✓	
	_swapV2WithPath	Internal	✓	
	_swapCamelotV3	Internal	✓	
	_swapCroc	Internal	✓	
	_swapCrocWithPath	Internal	✓	
	_getQuoteCroc	Internal		
	_getQuoteCrocWithPath	Internal		
	_getQuoteV3	Internal		
	_getQuoteV3WithPath	Internal		
	_getQuoteV2	Internal		
	_getQuoteV2WithPath	Internal		
LpRouter	Implementation	ILpRouter		
		Public	✓	-
	setGovernance	Public	✓	onlyGovernance

	setDefaultDex	External	✓	onlyGovernance
	setRouter	External	✓	onlyGovernance
	addLiquidityWithDefaultDex	External	✓	-
	addLiquidity	Public	✓	-
	removeLiquidityWithDefaultDex	External	✓	-
	removeLiquidity	Public	✓	-
	_addLiquidityCrocSwap	Internal	✓	
	_removeLiquidityCrocSwap	Internal	✓	
StrategyBase	Implementation	IStrategy		
		Public	✓	-
	balanceOfAsset	Public		-
	balanceOfPool	Public		-
	balanceOf	Public		-
	_swapBGTToAsset	Internal	✓	
	whitelistHarvester	External	✓	-
	revokeHarvester	External	✓	-
	setWithdrawalDevFundFee	External	✓	onlyTimelock
	setWithdrawalTreasuryFee	External	✓	onlyTimelock
	setPerformanceDevFee	External	✓	onlyTimelock
	setPerformanceTreasuryFee	External	✓	onlyTimelock
	setStrategist	External	✓	onlyGovernance

	setGovernance	External	✓	onlyGovernance
	setTimelock	External	✓	onlyTimelock
	setController	External	✓	onlyTimelock
	deposit	Public	✓	-
	harvest	Public	✓	-
	_withdrawSome	Internal	✓	
	withdraw	External	✓	onlyController
	withdraw	External	✓	onlyController
	withdrawForSwap	External	✓	onlyController
	withdrawAll	External	✓	onlyController
	_withdrawAll	Internal	✓	
	execute	Public	Payable	onlyTimelock
	_distributePerformanceFeesBasedAmountAndDeposit	Internal	✓	
SteerStrategy	Implementation	StrategyBase		
		Public	✓	StrategyBase
	deposit	Public	✓	-
	getHarvestable	External		-
	balanceOfPool	Public		-
	_withdrawSome	Internal	✓	
KodiakStrategy	Implementation	StrategyBase		

		Public	✓	StrategyBase
	deposit	Public	✓	-
	getHarvestable	External		-
	balanceOfPool	Public		-
	_withdrawSome	Internal	✓	
InfraredStrategy	Implementation	StrategyBase		
		Public	✓	StrategyBase
	deposit	Public	✓	-
	getHarvestable	External		-
	harvest	Public	✓	-
	_swapAndAddLiquidity	Internal	✓	
	balanceOfPool	Public		-
	_withdrawSome	Internal	✓	
	setRewardTokensLength	External	✓	onlyGovernance
ArberaStrategy	Implementation	StrategyBase		
		Public	✓	StrategyBase
	deposit	Public	✓	-
	getHarvestable	External		-
	balanceOfPool	Public		-
	_withdrawSome	Internal	✓	


VaultAndControllerFactory	Implementation	IVaultAndControllerFactory		
		Public	✓	-
	setDev	External	✓	onlyDev
	setStrategyFactory	External	✓	onlyDev
	createVaultAndController	External	✓	onlyStrategyFactory
StrategyFactoryBase	Implementation	IStrategyFactory		
		Public	✓	-
	setDev	External	✓	onlyDev
	setVaultAndControllerFactory	External	✓	onlyDev
	_createControllerAndVault	Internal	✓	
	_setupVault	Internal	✓	
	createVault	External	✓	onlyDev onlyNewAsset
	_deployStrategyByteCode	Internal	✓	
	setVaultData	External	✓	onlyDev
SteerFactory	Implementation	StrategyFactoryBase		
		Public	✓	StrategyFactoryBase
	createVaultWithParams	External	✓	onlyDev onlyNewAsset
KodiakFactory	Implementation	StrategyFactoryBase		

		Public	✓	StrategyFactory Base
	createVaultWithParams	External	✓	onlyDev onlyNewAsset
ArberaFactory	Implementation	StrategyFact oryBase		
		Public	✓	StrategyFactory Base
	createVaultWithParams	External	✓	onlyDev onlyNewAsset
Controller	Implementation	IController		
		Public	✓	-
	setDevFund	Public	✓	onlyGovernanc e
	setTreasury	Public	✓	onlyGovernanc e
	setStrategist	Public	✓	onlyGovernanc e
	setGovernance	Public	✓	onlyGovernanc e
	setTimelock	Public	✓	onlyTimelock
	setVault	Public	✓	onlyStrategist
	approveStrategy	Public	✓	onlyTimelock
	revokeStrategy	Public	✓	onlyGovernanc e
	setStrategy	Public	✓	onlyStrategist
	earn	Public	✓	-
	balanceOf	External		-
	withdrawAll	Public	✓	onlyStrategist

	inCaseTokensGetStuck	Public	✓	onlyGovernance
	inCaseStrategyTokenGetStuck	Public	✓	onlyGovernance
	withdraw	Public	✓	onlyVault


Inheritance Graph

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

 Inheritance Graph.png

Flow Graph

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

 Flow 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