



# Cyberscope

## Audit Report

# BeraTrax

February 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

<b>Table of Contents</b>	<b>1</b>
<b>Risk Classification</b>	<b>3</b>
<b>Review</b>	<b>4</b>
Audit Updates	4
Source Files	4
<b>Overview</b>	<b>6</b>
Controller	6
Factories	6
Strategies	6
Vault	7
Zappers	7
Disclaimer	7
<b>Findings Breakdown</b>	<b>8</b>
<b>Diagnostics</b>	<b>9</b>
ELFM - Exceeds Fees Limit	10
Description	10
Recommendation	11
CCR - Contract Centralization Risk	12
Description	12
Recommendation	13
MN - Misspelled Naming	14
Description	14
Recommendation	16
PLPI - Potential Liquidity Provision Inadequacy	17
Description	17
Recommendation	20
USV - Uninitialized State Variable	21
Description	21
Recommendation	21
UTPD - Unverified Third Party Dependencies	22
Description	22
Recommendation	23
L02 - State Variables could be Declared Constant	24
Description	24
Recommendation	24
L04 - Conformance to Solidity Naming Conventions	25
Description	25
Recommendation	25
L09 - Dead Code Elimination	26

Description	26
Recommendation	26
L14 - Uninitialized Variables in Local Scope	27
Description	27
Recommendation	27
L15 - Local Scope Variable Shadowing	28
Description	28
Recommendation	28
L16 - Validate Variable Setters	29
Description	29
Recommendation	30
L17 - Usage of Solidity Assembly	31
Description	31
Recommendation	32
<b>Functions Analysis</b>	<b>33</b>
<b>Inheritance Graph</b>	<b>49</b>
<b>Summary</b>	<b>50</b>
<b>Disclaimer</b>	<b>51</b>
<b>About Cyberscope</b>	<b>52</b>

## 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  <a href="https://github.com/cyberscope-io/audits/blob/main/c5-btx/v1/audit.pdf">https://github.com/cyberscope-io/audits/blob/main/c5-btx/v1/audit.pdf</a>
Corrected Phase 2	31 Jan 2025

## Source Files

Filename	SHA256
zappers/ZapperBase.sol	ff654b6361ffae91da157ee7a16404b59900 3c5b979a7308a5ac4d65986b3496
zappers/SteerZapper.sol	2b2be5b543a1ffec368c7c8f5e81b41f9066 8a8789023729a4066893804d1088
zappers/KodiakZapper.sol	e14d23df8ec057066500c3e95134e2be4f7 1bfa22a4fb870e48e989f2045db13
zappers/InfraredZapper.sol	01a2b5564680c7eae904d4af5fc1e56e7ad 5a2eae944dd26813c26da289550ea
zappers/ArberaZapper.sol	9faae00fe5c73ef26c3fb348635766666ce1 985abfd5dea66972136ff3adca4d
vaults/Vault.sol	81307b538144b4fd72ba6c91bdc6a16918 6bc828e5bd723b0a92cc9b023c7fa9
utils/SwapRouter.sol	05047b0402b67ceb2a6151c7bf2285e286 d27594ed96f9e28cc4509fcd7319d6

<b>utils/LpRouter.sol</b>	bac53bbb10cf6a14e4857d6521bf5b1771 cf837ad505472f544d4d1381c4b158
<b>strategies/StrategyBase.sol</b>	32683332fc6eb7eabfab3a0a39d82cf73fd 8e0c067d48fd904c56acfd45ee36
<b>strategies/SteerStrategy.sol</b>	3ad54d64918c52ca51d99927de12d2aea0 fa7bf7d78758fc35117622e6a31c99
<b>strategies/KodiakStrategy.sol</b>	289809d4c4937f7d0573528ad5f38ae89b 3e440680c95ca69985e61ce938b234
<b>strategies/InfraredStrategy.sol</b>	11555d5d20a39ce4f492600fcd231c1b739 b118273e4e6bd73850c10dee3552f
<b>strategies/ArberaStrategy.sol</b>	d780c7c63a8ea516fe926bc765fe321e9db 4dc6a01c77d4875769178b965ef56
<b>factories/VaultFactory.sol</b>	e9230c9b34c80543b9eaa2e6906d90fa43 bfa12346775f8b1c8c1a5fd0ff32c4
<b>factories/ControllerFactory.sol</b>	eed63c88d2896f9f85db51e50df0024b389 0fd74fb834d141c03852321d48cfc
<b>factories/StrategyFactoryBase.sol</b>	4c293174ad20551ba5d31c1273c818d61d 725c135ffab9e3e26641e786d6329c
<b>factories/SteerFactory.sol</b>	955b97383f87f936de1bc4584851a61959f 37ce3a3f64eb9eb68720ba94c083a
<b>factories/KodiakFactory.sol</b>	91b8e794676da1ca5f26e9fdc7697becc70 087c20f625d7a8de8e7386324c164
<b>factories/InfraredFactory.sol</b>	735682af2ef3b5073d830e12eced2635496 5504b50f6b0912fa2b8c9412d73ca
<b>factories/ArberaFactory.sol</b>	4e23d79a16f373a1bff533dd65b206e10e0 1f34d9b6c8c8b0abbcc5a907a1e05
<b>controllers/Controller.sol</b>	d1dbd00e30ebfc4ce21b48d43c5799d659 3c2b70a878358068e759a9d468616b

## 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

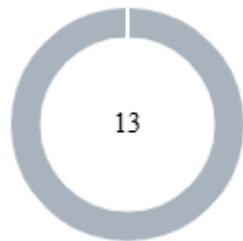
The Zappers are designed to enhance user convenience by automating asset conversions and liquidity provision for deposits and withdrawals. They allow users to interact with Vaults using various tokens, handling the necessary swaps, liquidity additions, and token approvals behind the scenes. Zappers streamline 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.

## Disclaimer

@spherex-xyz/contracts, being an external source, are considered out of scope for this review and will not be analyzed as part of this assessment.



## Findings Breakdown



● Critical	0
● Medium	0
● Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	13	0	0	0

# Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	ELFM	Exceeds Fees Limit	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	MN	Misspelled Naming	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	USV	Uninitialized State Variable	Unresolved
●	UTPD	Unverified Third Party Dependencies	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	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

## ELFM - Exceeds Fees Limit

Criticality	Minor / Informative
Location	strategies/StrategyBase.sol#L27
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.

```
function setPerformanceDevFee(uint16 fee) external onlyTimelock
sphereXGuardExternal(0x39189706) {
    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 sphereXGuardExternal(0x76672d92) {
    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.

## CCR - Contract Centralization Risk

<b>Criticality</b>	Minor / Informative
<b>Location</b>	controllers/Controller.sol#L154,163,172,217 vaults/Vault.sol#L195 factories/StrategyFactoryBase.sol#L76,85,94,103,264
<b>Status</b>	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) public
onlyGovernance sphereXGuardPublic(0xefc74f14, 0xab033ea9) {}

function setTimelock(address timelockAddress) public
onlyTimelock sphereXGuardPublic(0xdecf35e6, 0xbdacb303) {}

function setController(address controllerAddress) external
onlyTimelock sphereXGuardExternal(0x431250ca) {}

function setStrategist(address strategistAddress) public
onlyGovernance sphereXGuardPublic(0x15f23c15, 0xc7b9d530) {}

function setStrategy(
    address asset,
    address strategy
) public nonReentrant onlyStrategist
sphereXGuardPublic(0x632853be, 0x72cb5d97) {}

function setSphereXEngine(address sphereXEngineAddress)
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 setDev(address devAddress) external onlyDev {}
```

## 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.

## MN - Misspelled Naming

<b>Criticality</b>	Minor / Informative
<b>Location</b>	utils/SwapRouter#L165
<b>Status</b>	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 sphereXGuardExternal(0x1a19f525) {  
  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 (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  
        })  
      );  
    }  
  }  
  
  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.

## PLPI - Potential Liquidity Provision Inadequacy

<b>Criticality</b>	Minor / Informative
<b>Location</b>	utils/SwapRouter.sol#L532
<b>Status</b>	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.

## USV - Uninitialized State Variable

Criticality	Minor / Informative
Location	vaults/Vault.sol#L50
Status	Unresolved

### Description

In the `Vault` contract, there is an address state variable declared as `feeRecipient` but there is no functionality that provides a valid address for it. This results in the `feeRecipient` to always be `address(0)`.

```
address public feeRecipient;
```

### Recommendation

To prevent `feeRecipient` from defaulting to `address(0)`, if not intended, it is recommended to add the functionality needed to set it with a valid address.

## UTPD - Unverified Third Party Dependencies

Criticality	Minor / Informative
Location	strategies/StrategyBase.sol#L187 strategies/InfraredStrategy.sol#L79 factories/StrategyFactoryBase.sol#L168
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);
}
```

```
address public sphereXEngine;
...
ISphereXEngine(sphereXEngine).addAllowedSenderOnChain(address(controller));
ISphereXEngine(sphereXEngine).addAllowedSenderOnChain(address(vault)
);
```

## 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.



## L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	vaults/Vault.sol#L50
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.

```
address public feeRecipient
```

### Recommendation

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

## L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	strategies/ArberaStrategy.sol#L128
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>.

## L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	zappers/ZapperBase.sol#L203,232,249
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 sphereXGuardInternal(0x367a5b5d) {  
    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	zappers/KodiakZapper.sol#L202
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.

```
Amounts memory amounts
```

### 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

<b>Criticality</b>	Minor / Informative
<b>Location</b>	zappers/SteerZapper.sol#L34,35,36,37,38,39,40 zappers/KodiakZapper.sol#L34,36,37,38,39,40 interfaces/IController.sol#L105,108,111 zappers/InfraredZapper.sol#L22,23,24,25,26,27,28 zappers/ArberaZapper.sol#L26,27,28,29,30,31,32
<b>Status</b>	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.

## L16 - Validate Variable Setters

<b>Criticality</b>	Minor / Informative
<b>Location</b>	zappers/ZapperBase.sol#L119,184 factories/VaultFactory.sol#L29,60 vaults/Vault.sol#L76,77,78,176,187,198 utils/SwapRouter.sol#L64,65,123 factories/StrategyFactoryBase.sol#L79,88 strategies/StrategyBase.sol#L260,269,278,287 utils/LpRouter.sol#L35,61 factories/ControllerFactory.sol#L29,64 controllers/Controller.sol#L139,148,157,166,175 strategies/ArberaStrategy.sol#L63,130
<b>Status</b>	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
feeRecipient = newRecipient
dev = devAddress
timelock = timelockAddress
controller = controllerAddress
wrappedNative = wrappedNativeAddress
sphereXEngine = sphereXEngineAddress
strategist = strategistAddress
devfund = devfundAddress
treasury = treasuryAddress
rewardToken = rewardTokenAddress
rewardToken = _rewardToken
```

## 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	factories/StrategyFactoryBase.sol#L309 strategies/StrategyBase.sol#L438
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)
    }
}
```

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

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

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



```
}
```

## Recommendation

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

# Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
<b>ZapperBase</b>	Implementation	SphereXProtected, ReentrancyGuard, IZapper		
		Public	✓	-
	setGovernance	External	✓	onlyGovernance
	setSwapRouter	External	✓	onlyGovernance sphereXGuardExternal
	setLpRouter	External	✓	onlyGovernance sphereXGuardExternal
	setStableCoin	External	✓	onlyGovernance sphereXGuardExternal
	setZapInFee	External	✓	onlyGovernance
	setZapOutFee	External	✓	onlyGovernance
	setFeeRecipient	External	✓	onlyGovernance
		External	Payable	-
	_revertAddressZero	Internal		
	_approveTokenIfNeeded	Internal	✓	sphereXGuardInternal
	_safeTransferFromTokens	Internal	✓	sphereXGuardInternal

	_divideAmountInRatio	Internal		
	_returnAssets	Internal	✓	sphereXGuardInternal
	_returnAsset	Internal	✓	sphereXGuardInternal
	_returnAssetWithAmount	Internal	✓	
	_transferFee	Internal	✓	
	swapToAssets	Public	✓	-
	swapFromAssets	Public	✓	-
	zapIn	External	Payable	nonReentrant sphereXGuardExternal
	zapOut	External	✓	nonReentrant sphereXGuardExternal
<b>VaultMigrator</b>	Implementation	Ownable		
		Public	✓	Ownable
	migrate	External	✓	onlyOwner sphereXGuardExternal
	rescueTokens	External	✓	onlyOwner sphereXGuardExternal
	rescueEth	External	✓	onlyOwner sphereXGuardExternal
<b>VaultFactory</b>	Implementation	SphereXProtectedBase, IVaultFactory		
		Public	✓	SphereXProtectedBase
	_revertAddressZero	Internal		

	setDev	External	✓	onlyDev sphereXGuardE xternal
	setWhitelistedStrategyFactory	External	✓	onlyDev sphereXGuardE xternal
	setSphereXProtected	Private	✓	sphereXGuardI nternal
	createVault	External	✓	onlyWhitelisted StrategyFactory sphereXGuardE xternal
<b>Vault</b>	Implementation	SphereXProt ected, ReentrancyG uard, ERC4626, IVault		
		Public	✓	ERC4626 ERC20
	_revertOnlyGovernance	Internal		
	_revertOnlyTimelock	Internal		
	_revertOnlyController	Internal		
	_revertAddressZero	Internal		
	decimals	Public		-
	setMin	External	✓	onlyGovernanc e sphereXGuardE xternal
	setDepositFee	External	✓	onlyGovernanc e
	setWithdrawFee	External	✓	onlyGovernanc e
	setGovernance	External	✓	onlyGovernanc e sphereXGuardE xternal

	setTimelock	External	✓	onlyTimelock sphereXGuardE xternal
	setController	External	✓	onlyTimelock sphereXGuardE xternal
	available	Public		-
	totalAssets	Public		-
	earn	Public	✓	nonReentrant sphereXGuardP ublic
	harvest	External	✓	nonReentrant onlyController sphereXGuardE xternal
	_withdraw	Internal	✓	sphereXGuardI nternal
	_deposit	Internal	✓	sphereXGuardI nternal
	deposit	Public	✓	nonReentrant sphereXGuardP ublic
	redeem	Public	✓	nonReentrant sphereXGuardP ublic
<b>SwapRouter</b>	Implementation	SphereXProt ected, ReentrancyG uard, ISwapRouter		
		Public	✓	-
	validateSwapParams	Internal		
	validateSwapWithPathParams	Internal		
	setGovernance	Public	✓	onlyGovernanc e sphereXGuardP ublic
	setDefaultDex	External	✓	onlyGovernanc e

				sphereXGuardExternal
	setRouter	External	✓	onlyGovernance sphereXGuardExternal
	setFactory	External	✓	onlyGovernance sphereXGuardExternal
	setCrocQuery	External	✓	onlyGovernance sphereXGuardExternal
	setSwapRoute	External	✓	onlyGovernance sphereXGuardExternal
	swapWithDefaultDex	External	✓	resetPathLength sphereXGuardExternal
	swap	Public	✓	nonReentrant sphereXGuardPublic
	swapWithPathWithDefaultDex	Public	✓	nonReentrant sphereXGuardPublic
	swapWithPath	Public	✓	nonReentrant sphereXGuardPublic
	getQuoteWithDefaultDex	External		-
	getQuote	Public		-
	getQuoteWithPathWithDefaultDex	External		-
	getQuoteWithPath	Public		-
	_revertAddressZero	Internal		
	_revertZeroAmount	Internal		
	_revertInvalidPathLength	Internal		
	_findMostLiquidV3Pool	Internal		

	_findMostLiquidCrocPool	Internal		
	_swapWithRoute	Internal	✓	sphereXGuardInternal
	_swapV3	Internal	✓	sphereXGuardInternal
	_swapV3WithPath	Internal	✓	sphereXGuardInternal
	_swapV2	Internal	✓	sphereXGuardInternal
	_swapV2WithPath	Internal	✓	sphereXGuardInternal
	_swapCamelotV3	Internal	✓	sphereXGuardInternal
	_swapCroc	Internal	✓	sphereXGuardInternal
	_swapCrocWithPath	Internal	✓	sphereXGuardInternal
	_getQuoteCroc	Internal		
	_getQuoteCrocWithPath	Internal		
	_getQuoteV3	Internal		
	_getQuoteV3WithPath	Internal		
	_getQuoteV2	Internal		
	_getQuoteV2WithPath	Internal		
<b>StrategyFactory Base</b>	Implementation	IStrategyFactory		
		Public	✓	-
	revertOnlyDev	Private		
	_revertAddressZero	Internal		
	setDev	External	✓	onlyDev
	setSphereXEngine	External	✓	onlyDev

	setVaultFactory	External	✓	onlyDev
	setControllerFactory	External	✓	onlyDev
	_setAddressAsSpherexProtected	Internal	✓	
	_createControllerAndVault	Internal	✓	
	_setupVault	Internal	✓	
	revertIfNonInitializedParams	Private		
	_encodeParamsAndController	Private		
	createVault	External	✓	onlyDev onlyNewAsset
	_deployStrategyByteCode	Internal	✓	
<b>StrategyBase</b>	Implementation	SphereXProtected, ReentrancyGuard, IStrategy		
		Public	✓	-
		External	Payable	-
	balanceOfAsset	Public		-
	balanceOfPool	Public		-
	balanceOf	Public		-
	_revertAddressZero	Internal		
	_revertOnlyGovernance	Internal		
	_revertOnlyTimelock	Internal		
	_revertOnlyController	Internal		
	_revertOnlyBenevolent	Internal		
	_swapBGTToAsset	Internal	✓	sphereXGuardInternal



	whitelistHarvester	External	✓	sphereXGuardExternal
	revokeHarvester	External	✓	sphereXGuardExternal
	setWithdrawalDevFundFee	External	✓	onlyTimelock sphereXGuardExternal
	setWithdrawalTreasuryFee	External	✓	onlyTimelock sphereXGuardExternal
	setPerformanceDevFee	External	✓	onlyTimelock sphereXGuardExternal
	setPerformanceTreasuryFee	External	✓	onlyTimelock sphereXGuardExternal
	setStrategist	External	✓	onlyGovernance sphereXGuardExternal
	setGovernance	External	✓	onlyGovernance sphereXGuardExternal
	setTimelock	External	✓	onlyTimelock sphereXGuardExternal
	setController	External	✓	onlyTimelock sphereXGuardExternal
	setSwapRouter	External	✓	onlyGovernance sphereXGuardExternal
	setLpRouter	External	✓	onlyGovernance sphereXGuardExternal
	setZapper	External	✓	onlyGovernance sphereXGuardExternal
	deposit	Public	✓	-
	getHarvestable	External		-

	harvest	Public	✓	nonReentrant onlyBenevolent sphereXGuardP ublic
	_withdrawSome	Internal	✓	
	withdraw	External	✓	nonReentrant onlyController sphereXGuardE xternal
	withdraw	External	✓	nonReentrant onlyController sphereXGuardE xternal
	withdrawForSwap	External	✓	nonReentrant onlyController sphereXGuardE xternal
	withdrawAll	External	✓	nonReentrant onlyController sphereXGuardE xternal
	_withdrawAll	Internal	✓	sphereXGuardI nternal
	execute	Public	Payable	onlyTimelock sphereXGuardP ublic
	_distributePerformanceFeesBasedAmou ntAndDeposit	Internal	✓	sphereXGuardI nternal
<b>SteerZapper</b>	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	sphereXGuardP ublic
	swapFromAssets	Public	✓	sphereXGuardP ublic
	_getAmountsForLiquidity	Internal		
<b>SteerStrategy</b>	Implementation	StrategyBas e		

		Public	✓	StrategyBase
	deposit	Public	✓	sphereXGuardPublic
	balanceOfPool	Public		-
	_withdrawSome	Internal	✓	sphereXGuardInternal
<b>SteerFactory</b>	Implementation	StrategyFactoryBase		
		Public	✓	StrategyFactoryBase
	createVaultWithParams	External	✓	onlyDev onlyNewAsset
<b>LpRouter</b>	Implementation	SphereXProtected, ReentrancyGuard, ILpRouter		
		Public	✓	-
	setGovernance	Public	✓	onlyGovernance sphereXGuardPublic
	setDefaultDex	External	✓	onlyGovernance sphereXGuardExternal
	setRouter	External	✓	onlyGovernance sphereXGuardExternal
	addLiquidityWithDefaultDex	External	✓	nonReentrant sphereXGuardExternal
	addLiquidity	Public	✓	nonReentrant sphereXGuardPublic
	removeLiquidityWithDefaultDex	External	✓	nonReentrant sphereXGuardExternal

	removeLiquidity	Public	✓	nonReentrant sphereXGuardP ublic
	_revertAddressZero	Internal		
	_encodeDataForAddition	Private		
	_encodeDataForRemoval	Private		
	_addLiquidityCrocSwap	Internal	✓	sphereXGuardI nternal
	_removeLiquidityCrocSwap	Internal	✓	sphereXGuardI nternal
<b>KodiakZapper</b>	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	sphereXGuardP ublic
	swapFromAssets	Public	✓	sphereXGuardP ublic
	_getTokenBalances	Internal		
	_getKodiakVaultMintAmount	Internal		
	_getAmountsForLiquidity	Internal		
	_addLiquidity	Internal	✓	sphereXGuardI nternal
<b>KodiakStrategy</b>	Implementation	StrategyBase		
		Public	✓	StrategyBase
	deposit	Public	✓	sphereXGuardP ublic
	balanceOfPool	Public		-
	_withdrawSome	Internal	✓	sphereXGuardI nternal

<b>KodiakFactory</b>	Implementation	StrategyFactoryBase		
		Public	✓	StrategyFactoryBase
	createVaultWithParams	External	✓	onlyDev onlyNewAsset
<b>InfraredZapper</b>	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	sphereXGuardPublic
	swapFromAssets	Public	✓	sphereXGuardPublic
<b>InfraredStrategy</b>	Implementation	StrategyBase		
		Public	✓	StrategyBase
	deposit	Public	✓	nonReentrant sphereXGuardPublic
	getHarvestable	External		-
	harvest	Public	✓	onlyBenevolent sphereXGuardPublic
	balanceOfPool	Public		-
	_withdrawSome	Internal	✓	sphereXGuardInternal
	setRewardTokensLength	External	✓	onlyGovernance sphereXGuardExternal
<b>InfraredFactory</b>	Implementation	StrategyFactoryBase		
		Public	✓	StrategyFactoryBase

	deployStrategyWithParamsAndController	Internal	✓	
	createVaultWithParams	External	✓	onlyDev onlyNewAsset
<b>ControllerFactory</b>	Implementation	SphereXProtectedBase, IControllerFactory		
		Public	✓	SphereXProtectedBase
	_revertAddressZero	Internal		
	setDev	External	✓	onlyDev sphereXGuardExternal
	setWhitelistedStrategyFactory	External	✓	onlyDev sphereXGuardExternal
	setSphereXProtected	Private	✓	sphereXGuardInternal
	createController	External	✓	onlyWhitelisted StrategyFactory sphereXGuardExternal
<b>Controller</b>	Implementation	SphereXProtected, ReentrancyGuard, IController		
		Public	✓	-
	_revertAddressZero	Internal		
	_revertOneAddressZero	Internal		
	_revertOnlyGovernance	Internal		
	_revertOnlyStrategist	Internal		
	_revertOnlyTimelock	Internal		

	setDevFund	Public	✓	onlyGovernance sphereXGuardPublic
	setTreasury	Public	✓	onlyGovernance sphereXGuardPublic
	setStrategist	Public	✓	onlyGovernance sphereXGuardPublic
	setGovernance	Public	✓	onlyGovernance sphereXGuardPublic
	setTimelock	Public	✓	onlyTimelock sphereXGuardPublic
	setVault	Public	✓	onlyStrategist sphereXGuardPublic
	approveStrategy	Public	✓	onlyTimelock sphereXGuardPublic
	revokeStrategy	Public	✓	onlyGovernance sphereXGuardPublic
	setStrategy	Public	✓	nonReentrant onlyStrategist sphereXGuardPublic
	earn	Public	✓	nonReentrant sphereXGuardPublic
	balanceOf	External		-
	withdrawAll	Public	✓	nonReentrant onlyStrategist sphereXGuardPublic
	inCaseTokensGetStuck	Public	✓	onlyGovernance sphereXGuardPublic


	inCaseStrategyTokenGetStuck	Public	✓	onlyGovernance sphereXGuardPublic
	withdraw	Public	✓	nonReentrant onlyVault sphereXGuardPublic
<b>ArberaZapper</b>	Implementation	ZapperBase		
		Public	✓	ZapperBase
	swapToAssets	Public	✓	sphereXGuardPublic
	swapFromAssets	Public	✓	sphereXGuardPublic
	swapToAssetsWithBond	Public	✓	sphereXGuardPublic
	swapFromAssetsWithBond	Public	✓	sphereXGuardPublic
	_getAmounts	Internal		
	zapIn	External	Payable	nonReentrant sphereXGuardExternal
	zapInWithBond	External	Payable	nonReentrant sphereXGuardExternal
	zapOutWithBond	External	✓	nonReentrant sphereXGuardExternal
<b>ArberaStrategy</b>	Implementation	StrategyBase		
		Public	✓	StrategyBase
	deposit	Public	✓	sphereXGuardPublic
	getHarvestable	External		-



	harvest	Public	✓	onlyBenevolent sphereXGuardP ublic
	balanceOfPool	Public		-
	_withdrawSome	Internal	✓	sphereXGuardI nternal
	setRewardToken	External	✓	onlyGovernanc e
<b>ArberaFactory</b>	Implementation	StrategyFact oryBase		
		Public	✓	StrategyFactory Base
	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](https://cyberscope.io)