// Private Smart Contract Security Assessment 01.22.2024 - 02.29.2024

Archimedes Finance - Multihopper

Archimedes Finance

HALBORN

Archimedes Finance - Multihopper - Archimedes Finance

Prepared by: HALBORN

Last Updated 04/03/2024

Date of Engagement by: January 22nd, 2024 - February 29th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
12	0	0	2	5	5

TABLE OF CONTENTS

6. Automated Testing

1. Introduction

The Multipool Hopper Smart Contracts are the infrastructure for automated AMM pool swapping. User can deposit base assets like ETH, USDC, WBTC and invest into different strategies.

\client engaged `Halborn` to conduct a security assessment on their smart contracts beginning on 2024-01-22 and ending on 2024-02-29. The security assessment was scoped to the smart contracts provided in the thisisarchimedes/MultiHopper GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

Assessment Summary

Halborn was provided 5 weeks for the engagement and assigned a full-time security engineer to review the security of the smart contracts in scope. The security team consists of a blockchain and smart contract security experts with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some security risks that should be addressed by the Archimedes Finance team. The main ones were the following:

- Allow the caller to configure the expected minimum amount of LP received in the `depositETH()` function.
- Transfer any unswapped tokens from the `GenericZapper` contract to the depositor.

Test Approach And Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (`solgraph`).

- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Static Analysis of security for scoped contract, and imported functions (`Slither`).
- Testnet deployment (`Foundry`, `Brownie`).

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

3. SCOPE

FILES AND REPOSITORY

(a) Repository: MultiHopper

(b) Assessed Commit ID: 6bfdc5b

(c) Items in scope:

- src/MultiPoolStrategyFactory.sol
- src/MultiPoolStrategy.sol
- src/AuraAdapterBase.sol
- src/AuraComposableStablePoolAdapter.sol
- src/ConvexPoolAdapter.sol
- src/AuraWeightedPoolAdapter.sol
- src/ERC4626UpgradeableModified.sol
- src/AuraStablePoolAdapter.sol
- src/zapper/ETHZapper.sol
- src/zapper/GenericZapper.sol
- src/zapper/USDCZapper.sol
- src/utils/BalancerErrors.sol
- src/utils/FixedPoint.sol
- src/utils/LogExpMath.sol
- src/utils/Math.sol
- src/UniswapV3Strategy.sol

Out-of-Scope: - External libraries and dependencies., - Economic attacks.

REMEDIATION COMMIT ID:

- 5bd84f45bd84f4
- 54792a754792a7

Out-of-Scope: New features/implementations after the remediation commit IDs.

4. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	5	5

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LACK OF SLIPPAGE CHECK IN DEPOSITETH() FUNCTION	MEDIUM	NOT APPLICABLE - 04/01/2024
LEFTOVER DEPOSIT AMOUNT IS NOT TRANSFERRED BACK IN GENERICZAPPER	MEDIUM	SOLVED - 04/01/2024
FEE IS LOST IF FEE RECIPIENT IS NOT CONFIGURED	LOW	RISK ACCEPTED - 04/01/2024
MISSING UPPER BOUND CHECK FOR FEE PERCENTAGE	LOW	RISK ACCEPTED - 04/01/2024
LACK OF DISABLEINITIALIZERS CALL TO PREVENT UNINITIALIZED CONTRACTS	LOW	SOLVED - 04/01/2024
LACK OF VALIDATION IN SETFEE AND SETACCEPTEDSLIPPAGE FUNCTIONS	LOW	RISK ACCEPTED - 04/01/2024
SINGLE STEP OWNERSHIP TRANSFER PROCESS	LOW	RISK ACCEPTED - 04/01/2024
MISTAKENLY SENT ERC20 TOKENS CAN NOT RESCUED FROM THE ZAPPER CONTRACTS	INFORMATIONAL	ACKNOWLEDGED - 04/01/2024
INCOMPATIBILITY WITH NON-STANDARD TOKENS	INFORMATIONAL	ACKNOWLEDGED - 04/01/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISSING EVENTS FOR CONTRACT OPERATIONS	INFORMATIONAL	ACKNOWLEDGED - 04/01/2024
MISSING ZERO ADDRESS CHECKS	INFORMATIONAL	ACKNOWLEDGED - 04/01/2024
GAS INEFFICENT FOR LOOPS	INFORMATIONAL	ACKNOWLEDGED - 04/01/2024

5. FINDINGS & TECH DETAILS

5.1 LACK OF SLIPPAGE CHECK IN DEPOSITETH() FUNCTION

// MEDIUM

Description

It was identified that the `depositETH()` function of the `ETHZapper` contract is missing the parameter controlling the expected minimum amount of LP received.

Code Location

Input
src/zapper/ETHZapper.sol

```
34      function depositETH(
35          address receiver,
36          address strategyAddress
37      )
38          external
39          payable
40          nonReentrant
41          returns (uint256 shares)
42      {
43          if (!strategyUsesWETH(strategyAddress)) revert StrategyAssetNotWE
44          if (receiver == address(0)) revert ReceiverIsZeroAddress();
45
46          if (msg.value == 0) revert EmptyInput();
47          IMultiPoolStrategy multipoolStrategy = IMultiPoolStrategy(strateg
48          if (multipoolStrategy.paused()) revert StrategyPaused();
49
50          uint256 amountETH = msg.value;
51
52          // wrap ether and then call deposit
53          IWETH(payable(WETH_ADDRESS)).deposit{ value: amountETH }();
54
55          //// we need to approve the strategy to spend our WETH
56          SafeERC20Upgradeable.safeApprove(IERC20Upgradeable(multipoolStrat
57          SafeERC20Upgradeable.safeApprove(
58              IERC20Upgradeable(multipoolStrategy.asset()), address(multipo
59          );
```

BVSS

O:A/AC:M/AX:L/C:N/I:N/A:N/D:N/Y:H/R:N/S:U (5.0)

Recommendation

It is recommended to allow the caller to configure the expected minimum amount of LP received in the `depositETH()` function.

Remediation Plan

NOT APPLICABLE: The **Archimedes Finance team** stated that, this function only wraps ether to wrapped ether and deposit into our strategy, but our strategy does not implement any slippage logic since there is no such a thing.

5.2 LEFTOVER DEPOSIT AMOUNT IS NOT TRANSFERRED BACK IN GENERICZAPPER

// MEDIUM

Description

It is possible to deposit using generic ER20 tokens into the `MultiPoolStrategy` with the `GenericZapper` contract. The `GenericZapper` contract pulls the tokens from the caller and swaps them using the `LIFI DIAMOND` contract based on the `swapTx` parameter.
However, if the `swapTx` contains inaccurate data (e.g., the amount in `swapTx` is different from the `amount` function parameter), it is possible that not all tokens will be swapped to the underlying asset, and any leftover tokens will remain locked in the `GenericZapper` contract.

Code Location

The fee is lost if the `feeRecipient` address is not configured:
Input
src/zapper/GenericZapper.sol

```
29      function deposit(
30          uint256 amount,
31          address token,
32          uint256 toAmountMin,
33          address receiver,
34          address strategyAddress,
35          bytes calldata swapTx
36      )
37          external
38          returns (uint256 shares)
39      {
40          MultiPoolStrategy multiPoolStrategy = MultiPoolStrategy(strategyA
41
42          // check if the reciever is not zero address
43          if (receiver == address(0)) revert ZeroAddress();
44          // check if the amount is not zero
45          if (amount == 0) revert EmptyInput();
46
47          address underlyingAsset = multiPoolStrategy.asset();
48
49          // transfer tokens to this contract
50          uint256 underlyingBalanceBefore = IERC20(underlyingAsset).balance
51          SafeERC20.safeTransferFrom(IERC20(token), _msgSender(), address(t
52
```

```
        // swap for the underlying asset
        if (token != underlyingAsset) {
            SafeERC20.safeApprove(IERC20(token), LIFI_DIAMOND, 0);
            SafeERC20.safeApprove(IERC20(token), LIFI_DIAMOND, amount);
            // solhint-disable-next-line avoid-low-level-calls
            (bool success,) = LIFI_DIAMOND.call(swapTx);
            if (!success) revert SwapFailed();
        }

        uint256 underlyingBalanceAfter = IERC20(underlyingAsset).balanceO
        uint256 underlyingAmount = underlyingBalanceAfter - underlyingBal
```

Proof of Concept

The `LIFI_DIAMOND` contract only swapped half of the token because of the inaccurate `swapTx` parameter, and the leftover tokens remained locked in the contract:

```
test:  0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496
genericZapper:  0x8a72212eFe6Bc2636D1a79ff7f4a56B849599c7C
alice:  0xF1F6619B38A98d6De0800F1DefC0a6399eB6d30C
------------------
(, uint256 toAmountMin, bytes memory txData) = getQuoteLiFi(USDT, multiPoolStrategy.asset(), 500 * 10**6, address(genericZapper))
------------------
genericZapper.deposit(1000 * 10**6,USDT,toAmountMin,alice,address(multiPoolStrategy),concat(concat(txData, txData), txData))
DEBUG: token:  0xdAC17F958D2ee523a2206206994597C13D831ec7
DEBUG: underlyingAsset:  0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48
DEBUG: zapper token balance before swap:  1000000000
DEBUG: zapper underlyingAsset balance before swap:  0
DEBUG: zapper token balance after swap:  500000000
DEBUG: zapper underlyingAsset balance after swap:  497308333
DEBUG: final zapper token balance:  500000000
DEBUG: final zapper underlyingAsset balance:  0
alice share balance:  497308333
```

BVSS

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:H/Y:N/R:N/S:U (5.0)

Recommendation

It is recommended to transfer any unswapped tokens back to the caller.

Remediation Plan

SOLVED: The **Archimedes Finance team** solved this issue.

Remediation Hash

5bd84f485344aa2bdd9cbb771fbff6b192c8b85e

5.3 FEE IS LOST IF FEE RECIPIENT IS NOT CONFIGURED

// LOW

Description

The address of the fee recipient is stored in the `feeRecipient` state variable of the `MultiPoolStrategy` contract. However, it was identified that this variable is not initialized by default and needs to be configured manually using the `changeFeeRecipient()` function. If the address is not configured, the protocol fee is transferred to the default zero address and, therefore, lost.

Code Location

The fee is lost if the `feeRecipient` address is not conifugred:
Input
src/MultiPoolStrategy.sol

```
354     uint256 fee;
355     if (totalClaimed > 0) {
356       fee = (totalClaimed * feePercentage) / 10_000;
357       if (fee > 0) {
358         SafeERC20Upgradeable.safeTransfer(IERC20Upgradeable(asset()), fee
359       }
360     }
```

BVSS

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:N/Y:M/R:N/S:U (3.4)

Recommendation

It is recommended to configure the fee recipient in the `initialize()` function of the `MultiPoolStrategy` contract.
Otherwise, the protocol should not deduct the fee if the fee recipient is not configured.

Remediation Plan

RISK ACCEPTED: The **Archimedes Finance team** accepted the risk of this finding.

5.4 MISSING UPPER BOUND CHECK FOR FEE PERCENTAGE

// LOW

Description

The protocol fee percentage can be configured in the `changeFeePercentage()` function of the `MultiPoolStrategy` contract. However, it was identified that this function lacks an upper bound check, and therefore, it is possible to configure a higher fee than 100%.

Code Location

Input
src/MultiPoolStrategy.sol

```
495    /**
496     * @notice Change the fee percentage
497     * @param _feePercentage New fee percentage. 10000 = 100%
498     */
499    function changeFeePercentage(uint256 _feePercentage) external onlyOwner
500      feePercentage = _feePercentage;
501    }
```

BVSS

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:M/Y:N/R:N/S:U (3.4)

Recommendation

It is recommended to limit the maximum value of the fee percentage.

Remediation Plan

RISK ACCEPTED: The **Archimedes Finance team** accepted the risk of this finding.

5.5 LACK OF DISABLEINITIALIZERS CALL TO PREVENT UNINITIALIZED CONTRACTS

// LOW

Description

The `AuraAdapterBase` contract uses the `Initializable` module from OpenZeppelin, and the implementations of these contracts are not marked as initialized in the constructor. An uninitialized instance can be initialized by anyone to take over the contract. Even if it does not affect the proxy contracts directly, it is a good practice to initialize them to prevent any unseen vulnerabilities. [In the latest versions] (https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#initializing_the_implementation_contract), this is done by calling the `_disableInitializers` function in the constructor.

BVSS

AO:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation

It is recommended to include a constructor and call the `_disableInitializers()` function to mark the upgradeable contracts as initialized automatically when they are deployed.

Remediation Plan

SOLVED: The **Archimedes Finance team** solved this issue.

Remediation Hash

54792a7f31f6dbdf64e98ed405f46d9da5be35b8

5.6 LACK OF VALIDATION IN SETFEE AND SETACCEPTEDSLIPPAGE FUNCTIONS

// LOW

Description

The smart contract `UniswapV3Strategy` includes functions `setFee` and `setAcceptedSlippage` that allow the contract owner to update the fee percentage and the accepted slippage for transactions. However, there is no validation on the values that can be set for these parameters. This lack of validation can lead to scenarios where the fee is set to an unreasonable percentage (potentially 100% or higher) or the accepted slippage is set to an amount that could either entirely prevent trades from executing or expose users to high slippage costs. Such settings can significantly impact the integrity of the contract's operations and users' funds, posing a high risk.

BVSS

AO:S/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:N/S:U (2.5)

Recommendation

It is crucial to implement validation checks within both `setFee` and `setAcceptedSlippage` functions to ensure that the input values are within reasonable limits. For the `setFee` function, a maximum fee limit should be enforced (e.g., not exceeding a certain percentage, such as 10% or 20%). For the `setAcceptedSlippage`, a validation should ensure that slippage is within a range that protects users from excessive loss due to market movements but still allows transactions to be processed under normal conditions. Adding these validations improves the contract's security posture by preventing malicious or accidental settings that could adversely affect the contract's and its users' interests.

Remediation Plan

RISK ACCEPTED: The **Archimedes Finance team** accepted the risk of this finding.

5.7 SINGLE STEP OWNERSHIP TRANSFER PROCESS

// LOW

Description

Ownership of the contracts can be lost as the `MultiPoolStrategy`, `MultiPoolStrategyFactory` `USDCZapper` contracts are inherited from the `Ownable` and `OwnableUpgradeable` contracts, and their ownership can be transferred in a single-step process. The address the ownership is changed to should be verified to be active or willing to act as the owner.

BVSS

AO:S/AC:L/AX:L/C:N/I:N/A:C/D:N/Y:N/R:N/S:U (2.0)

Recommendation

It is recommended to use the `Ownable2Step` and `Ownable2StepUpgradeable` libraries from OpenZeppelin over `Ownable` and `OwnableUpgradeable`.

Remediation Plan

RISK ACCEPTED: The **Archimedes Finance team** accepted the risk of this finding.

5.8 MISTAKENLY SENT ERC20 TOKENS CAN NOT RESCUED FROM THE ZAPPER CONTRACTS

// INFORMATIONAL

Description

It was identified that the zapper contracts are missing functions to sweep/recover accidental ERC-20 and Ether transfers. The zapper contracts are not supposed to hold tokens or Ether balances. Therefore, accidentally sent ERC-20 tokens or Ether will be locked in these contracts.

BVSS

AO:A/AC:L/AX:H/C:N/I:N/A:N/D:M/Y:N/R:N/S:U (1.7)

Recommendation

Consider adding functions to sweep accidental ERC-20 and Ether transfers to the zapper contracts.

Remediation Plan

ACKNOWLEDGED: The **Archimedes Finance team** acknowledged this finding.

5.9 INCOMPATIBILITY WITH NON-STANDARD TOKENS

// INFORMATIONAL

Description

It was identified that the contracts assume that the `safeTransferFrom()` or `safeTransfer` calls will transfer the full amount of tokens. This may not be true if the tokens being transferred are fee-on-transfer tokens, causing the received amount to be lesser than the accounted amount. For example, DGX (Digix Gold Token) and CGT (CACHE Gold) tokens apply transfer fees, and the USDT (Tether) token also has a currently disabled fee feature.

It was also identified that the contracts assumes that its token balances will not change over time without any token transfers, which not be true if the tokens being transferred were deflationary/rebasing tokens. For example, the supply of AMPL (Ampleforth) tokens automatically increases or decreases every 24 hours to maintain the AMPL target price.

In these cases, the contract may not have the full token amounts, and the associated transfer functions may revert.

Code Location

For example, the `deposit()` function of the `ERC4626UpgradeableModified` contract assumes that the full `assets` amount will be transferred to the contract.

Input

src/ERC4626UpgradeableModified.sol

```
152     function deposit(uint256 assets, address receiver) public virtual ove
153         if (assets > maxDeposit(receiver)) {
154             revert DepositExceedsMax();
155         }
156 
157         uint256 shares = previewDeposit(assets);
158         _deposit(_msgSender(), receiver, assets, shares);
159 
160         return shares;
161     }
```

BVSS

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (1.7)

Recommendation

It is recommended to get the exact received amount of the tokens being transferred by calculating the difference of the token balance before and after the transfer and using it to update all the variables

correctly to make the contracts compatible with fee-on-transfer tokens.
It is also recommended that all tokens are thoroughly checked and tested before they are used to avoid tokens that are incompatible with the contracts.

Remediation Plan

ACKNOWLEDGED: The **Archimedes Finance team** acknowledged this finding.

5.10 MISSING EVENTS FOR CONTRACT OPERATIONS

// INFORMATIONAL

Description

It was identified that in the several high privileged functions do not emit any events in the contracts. As a result, blockchain monitoring systems might not be able to timely detect suspicious behaviors.

Code Location

An example admin function that does not emit an event:
Input
src/MultiPoolStrategy.sol

```
454    function setMonitor(address _monitor) external onlyOwner {
455      monitor = _monitor;
456    }
```

BVSS

AO:A/AC:L/AX:H/C:N/I:L/A:N/D:L/Y:L/R:N/S:U (1.2)

Recommendation

Adding events for all important operations is recommended to help monitor the contracts and detect suspicious behavior. A monitoring system that tracks relevant events would allow the timely detection of compromised system components.

Remediation Plan

ACKNOWLEDGED: The **Archimedes Finance team** acknowledged this finding.

5.11 MISSING ZERO ADDRESS CHECKS

// INFORMATIONAL

Description

It was identified that several address function parameters in the contracts lack zero address validation. Setting invalid values might result in loss of funds (e.g., fees) or cause some protocol operation to revert.

Code Location

The following functions are examples where zero address validation is missing:
`src/MultiPoolStrategyFactory.sol`

- constructor
- setMonitorAddress

`src/MultiPoolStrategy`

- addAdapter
- setMonitor
- changeFeeRecipient

BVSS

AO:A/AC:L/AX:H/C:N/I:L/A:N/D:L/Y:L/R:N/S:U (1.2)

Recommendation

Consider adding zero address validation for the address parameters of the functions.

Remediation Plan

ACKNOWLEDGED: The **Archimedes Finance team** acknowledged this finding.

5.12 GAS INEFFICENT FOR LOOPS

// INFORMATIONAL

Description

It was identified that postfix (e.g. `i++`) operators were used in several contracts to increment loop iterator variables. It is known that, in loops, using prefix operators (e.g. `++i`) costs less gas per iteration than postfix operators.
Furthermore, it was also identified that several loops are reading of the array length on each iteration.

Code Location

The following is an example from the `MultiPoolStrategy` contract:
Input
src/MultiPoolStrategy.sol

```
161     for (uint256 i = 0; i < _adapters.length; i++) {
162       bool isHealthy = IAdapter(_adapters[i]).isHealthy();
163       if (!isHealthy) revert AdapterNotHealthy();
164     }
```

Score

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation

Consider using prefix operation instead of postfix to increment the values of the `uint` loop iterator variables. Also, consider caching array lengths outside of loops, as long the size is not changed during the loop.

Remediation Plan

ACKNOWLEDGED: The **Archimedes Finance team** acknowledged this finding.

6. AUTOMATED TESTING

STATIC ANALYSIS REPORT

Description

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base. The security team assessed all findings identified by the Slither software, however, findings with severity `Information` and `Optimization` are not included in the below results for the sake of report readability.

Results

`src/MultiPoolStrategyFactory.sol`
Slither did not identify any vulnerabilities in the contract.
`src/MultiPoolStrategy.sol`

Slither results for MultiPoolStrategy.sol	
Finding	**Impact**
MultiPoolStrategy._withdrawFromAdapter(uint256,uint256,uint256) (src/MultiPoolStrategy.sol#226-261) performs a multiplication on the result of a division: - _lpAmount = (((_amount * 10 ** decimals()) / _adapterAssets) * lpBal) / (10 ** decimals()) (src/MultiPoolStrategy.sol#238)	Medium
MultiPoolStrategy._syncRewards(uint256) (src/MultiPoolStrategy.sol#368-390) performs a multiplication on the result of a division: - end = ((timestamp + rewardsCycleLength) / rewardsCycleLength) * rewardsCycleLength (src/MultiPoolStrategy.sol#378)	Medium
MultiPoolStrategy._withdrawFromAdapter(uint256,uint256,uint256) (src/MultiPoolStrategy.sol#226-261) uses a dangerous strict equality: - _assets == 0 (src/MultiPoolStrategy.sol#242)	Medium
MultiPoolStrategy._updateRewards() (src/MultiPoolStrategy.sol#395-402) uses a dangerous strict equality: - lastRewardAmount_ == 0 (src/MultiPoolStrategy.sol#397)	Medium
Reentrancy in MultiPoolStrategy.adjustOut(MultiPoolStrategy.Adjust[]) (src/MultiPoolStrategy.sol#271-286): External calls: - IAdapter(_adjustOuts[i].adapter).withdraw(_adjustOuts[i].amount, _adjustOuts[i].minReceive) (src/MultiPoolStrategy.sol#276) State variables written after the call(s): - lastAdjustOut = block.timestamp (src/MultiPoolStrategy.sol#283)MultiPoolStrategy.lastAdjustOut (src/MultiPoolStrategy.sol#28) can be used in cross function reentrancies: - MultiPoolStrategy.lastAdjustOut (src/MultiPoolStrategy.sol#28)	Medium

Finding	Impact
Reentrancy in MultiPoolStrategy.adjustIn(MultiPoolStrategy.Adjust[]) (src/MultiPoolStrategy.sol#288-306): External calls: - SafeERC20Upgradeable.safeTransfer(IERC20Upgradeable(asset()),_adjustIns[i].adapter,_adjustIns[i].amount) (src/MultiPoolStrategy.sol#294) - IAdapter(_adjustIns[i].adapter).deposit(_adjustIns[i].amount,_adjustIns[i].minReceive) (src/MultiPoolStrategy.sol#295) State variables written after the call(s): - lastAdjustIn = block.timestamp (src/MultiPoolStrategy.sol#303)MultiPoolStrategy.lastAdjustIn (src/MultiPoolStrategy.sol#26) can be used in cross function reentrancies: - MultiPoolStrategy.lastAdjustIn (src/MultiPoolStrategy.sol#26)	Medium
MultiPoolStrategy.doHardWork(address[],MultiPoolStrategy.SwapData[]).fee (src/MultiPoolStrategy.sol#354) is a local variable never initialized	Medium

src/AuraAdapterBase.sol

Slither results for AuraAdapterBase.sol	
Finding	**Impact**
AuraAdapterBase.withdraw(uint256,uint256) (src/AuraAdapterBase.sol#107-129) uses a dangerous strict equality: - lpBal == 0 (src/AuraAdapterBase.sol#119)	Medium
AuraAdapterBase.totalClaimable().i (src/AuraAdapterBase.sol#164) is a local variable never initialized	Medium
AuraAdapterBase.claim().i (src/AuraAdapterBase.sol#146) is a local variable never initialized	Medium
AuraAdapterBase.claim() (src/AuraAdapterBase.sol#135-153) ignores return value by auraRewardPool.getReward(address(this),true) (src/AuraAdapterBase.sol#136)	Medium
AuraAdapterBase.withdraw(uint256,uint256) (src/AuraAdapterBase.sol#107-129) ignores return value by auraRewardPool.withdrawAndUnwrap(_amount,false) (src/AuraAdapterBase.sol#112)	Medium

src/AuraComposableStablePoolAdapter.sol

Slither results for AuraComposableStablePoolAdapter.sol	
Finding	**Impact**
AuraComposableStablePoolAdapter.withdraw(uint256,uint256) (src/AuraComposableStablePoolAdapter.sol#87-119) uses a dangerous strict equality: - lpBal == 0 (src/AuraComposableStablePoolAdapter.sol#109)	Medium
AuraComposableStablePoolAdapter.underlyingBalance() (src/AuraComposableStablePoolAdapter.sol#23-62) uses a dangerous strict equality: - lpBal == 0 (src/AuraComposableStablePoolAdapter.sol#25)	Medium
AuraComposableStablePoolAdapter.withdraw(uint256,uint256).swap (src/AuraComposableStablePoolAdapter.sol#90) is a local variable never initialized	Medium
AuraComposableStablePoolAdapter.underlyingBalance().i (src/AuraComposableStablePoolAdapter.sol#33) is a local variable never initialized	Medium
AuraComposableStablePoolAdapter.deposit(uint256,uint256).swap (src/AuraComposableStablePoolAdapter.sol#69) is a local variable never initialized	Medium
AuraComposableStablePoolAdapter.withdraw(uint256,uint256).funds (src/AuraComposableStablePoolAdapter.sol#97) is a local variable never initialized	Medium
AuraComposableStablePoolAdapter.deposit(uint256,uint256).funds (src/AuraComposableStablePoolAdapter.sol#76) is a local variable never initialized	Medium
AuraComposableStablePoolAdapter.withdraw(uint256,uint256) (src/AuraComposableStablePoolAdapter.sol#87-119) ignores return value by vault.swap(swap,funds,_minReceiveAmount,block.timestamp + 20) (src/AuraComposableStablePoolAdapter.sol#105)	Medium
AuraComposableStablePoolAdapter.deposit(uint256,uint256) (src/AuraComposableStablePoolAdapter.sol#64-85) ignores return value by vault.swap(swap,funds,_minReceiveAmount,block.timestamp + 20) (src/AuraComposableStablePoolAdapter.sol#81)	Medium
AuraComposableStablePoolAdapter.withdraw(uint256,uint256) (src/AuraComposableStablePoolAdapter.sol#87-119) ignores return value by auraRewardPool.withdrawAndUnwrap(_amount,false) (src/AuraComposableStablePoolAdapter.sol#89)	Medium

Finding	Impact
AuraComposableStablePoolAdapter.deposit(uint256,uint256) (src/AuraComposableStablePoolAdapter.sol#64-85) ignores return value by IBooster(AURA_BOOSTER).deposit(auraPid,lpBal,true) (src/AuraComposableStablePoolAdapter.sol#83)	Medium

src/ConvexPoolAdapter.sol

Slither results for ConvexPoolAdapter.sol	
Finding	**Impact**
ConvexPoolAdapter.claim().i (src/ConvexPoolAdapter.sol#244) is a local variable never initialized	Medium
ConvexPoolAdapter.initialize(address,address,uint256,uint256,address,bool,bool,int128).i (src/ConvexPoolAdapter.sol#116) is a local variable never initialized	Medium
ConvexPoolAdapter.totalClaimable().i (src/ConvexPoolAdapter.sol#304) is a local variable never initialized	Medium
ConvexPoolAdapter.claim() (src/ConvexPoolAdapter.sol#233-251) ignores return value by convexRewardPool.getReward(address(this),true) (src/ConvexPoolAdapter.sol#234)	Medium
ConvexPoolAdapter.deposit(uint256,uint256) (src/ConvexPoolAdapter.sol#196-208) ignores return value by IBooster(CONVEX_BOOSTER).deposit(convexPid,curveLpBalance,true) (src/ConvexPoolAdapter.sol#206)	Medium
ConvexPoolAdapter.withdraw(uint256,uint256) (src/ConvexPoolAdapter.sol#210-231) ignores return value by convexRewardPool.withdrawAndUnwrap(_amount,false) (src/ConvexPoolAdapter.sol#212)	Medium

src/AuraWeightedPoolAdapter.sol

Slither results for AuraWeightedPoolAdapter.sol	
Finding	**Impact**
AuraWeightedPoolAdapter.underlyingBalance() (src/AuraWeightedPoolAdapter.sol#16-34) uses a dangerous strict equality: - lpBal == 0 (src/AuraWeightedPoolAdapter.sol#18)	Medium
End of table for AuraWeightedPoolAdapter.sol	

src/AuraStablePoolAdapter.sol

Slither results for AuraStablePoolAdapter.sol	
Finding	**Impact**
AuraStablePoolAdapter.underlyingBalance() (src/AuraStablePoolAdapter.sol#18-49) uses a dangerous strict equality: - lpBal == 0 (src/AuraStablePoolAdapter.sol#20)	Medium
AuraStablePoolAdapter.underlyingBalance().i (src/AuraStablePoolAdapter.sol#28) is a local variable never initialized	Medium
End of table for AuraStablePoolAdapter.sol	

src/zapper/ETHZapper.sol

Slither results for ETHZapper.sol	
Finding	**Impact**
ETHZapper.redeemETH(uint256,address,uint256,address) (src/zapper/ETHZapper.sol#115-139) sends eth to arbitrary user Dangerous calls: - address(address(receiver)).transfer(received) (src/zapper/ETHZapper.sol#136)	High
ETHZapper.withdrawETH(uint256,address,uint256,address) (src/zapper/ETHZapper.sol#75-104) sends eth to arbitrary user Dangerous calls: - address(address(receiver)).transfer(assets) (src/zapper/ETHZapper.sol#101)	High
End of table for ETHZapper.sol	

src/zapper/GenericZapper.sol

Slither results for GenericZapper.sol	
Finding	**Impact**
GenericZapper.deposit(uint256,address,uint256,address,address,bytes) (src/zapper/GenericZapper.sol#29-80) uses a dangerous strict equality: - underlyingAmount == 0 (src/zapper/GenericZapper.sol#65)	Medium

src/zapper/USDCZapper.sol

Slither results for USDCZapper.sol	
Finding	**Impact**
USDCZapper.constructor() (src/zapper/USDCZapper.sol#51-68) ignores return value by _supportedAssets.add(CRVFRAX) (src/zapper/USDCZapper.sol#58)	Medium
USDCZapper.removeAsset(address) (src/zapper/USDCZapper.sol#385-388) ignores return value by _supportedAssets.remove(asset) (src/zapper/USDCZapper.sol#386)	Medium
USDCZapper.constructor() (src/zapper/USDCZapper.sol#51-68) ignores return value by _supportedAssets.add(FRAX) (src/zapper/USDCZapper.sol#55)	Medium
USDCZapper.addAsset(address,USDCZapper.AssetInfo) (src/zapper/USDCZapper.sol#366-369) ignores return value by _supportedAssets.add(asset) (src/zapper/USDCZapper.sol#367)	Medium
USDCZapper.constructor() (src/zapper/USDCZapper.sol#51-68) ignores return value by _supportedAssets.add(USDT) (src/zapper/USDCZapper.sol#53)	Medium
USDCZapper.constructor() (src/zapper/USDCZapper.sol#51-68) ignores return value by _supportedAssets.add(CRV) (src/zapper/USDCZapper.sol#57)	Medium
USDCZapper.constructor() (src/zapper/USDCZapper.sol#51-68) ignores return value by _supportedAssets.add(DAI) (src/zapper/USDCZapper.sol#54)	Medium
End of table for USDCZapper.sol	

Results Summary

The findings obtained as a result of the Slither scan were reviewed:

- The `lack of zero-check on`, `should emit an event` and `has external calls inside a loop` informational findings were omitted from Slither's results and added to the report.
- The `uses timestamp for comparisons` and `has external calls inside loop` findings were omitted from Slither 's results to improve readability. However, these findings were manually reviewed and determined false-positves.
- The identified `reentrancy`, `uses a dangerous strict equality`, `local variable never initialized`, `sends eth to arbitrary user` and `ignores return value` vulnerabilities were determined false-positives.

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.