Three Sigma

Code Audit

ClipFinance

Clip Finance Secure, Automated Yield Solutions

Disclaimer

Code Audit

Clip Finance Secure, Automated Yield Solutions

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Clip Finance Secure, Automated Yield Solutions

Table of Contents

Summary

Code Audit

Clip Finance Secure, Automated Yield Solutions

Summary

Three Sigma Labs audited Clip Finance in a 2 person week engagement. The audit was conducted from 25/03/2024 to 1/04/2024. It focused on the upgrade of the strategy **`PCLBaseSwapInside`**.

Protocol Description

Clip Finance's modular smart contracts execute stablecoin yield farming strategies and return profits to Clip's users. All strategies go through a risk scoring procedure and are rigorously tested by the Clip Finance contributors before implementation.

These smart contracts are pieces of code that autonomously execute yield farming actions by interacting with the relevant external protocols. The strategies auto-compound token rewards to maximize profits. Despite the automated nature of Clip's strategies, users can withdraw their initial investment and profits from the protocol at **any** time.

Clip Finance is initially building on the Binance Smart Chain (BSC) but will be implementing yield farming strategies across 10+ leading blockchains.

Scope

Code Audit

Clip Finance Secure, Automated Yield Solutions

Scope

```
├── liquidityManagement
│   ├── PCLPoolHelper.sol
│   ├── PHyperLPoolSwapInside.sol
│   └── abstract
│       └── PHyperLPoolStorageSwapInside.sol
└── strategies
    └── PCLBaseSwapInside.sol
```

Assumptions

The admin roles in the protocol are trusted and the external libraries are considered secure.

Methodology

Code Audit

Clip Finance Secure, Automated Yield Solutions

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Clip Finance Secure, Automated Yield Solutions

Project Dashboard

Application Summary

Name	Clip Finance
Commit	07165c8876494564666e06d5e8a6b24c7e098465
Language	Solidity
Platform	Linea

Engagement Summary

Timeline	25/03/2024 to 1/04/2024
№ of Auditors	2
Review Time	2 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	2	2	0
Medium	1	1	0
Low	7	6	1
None	1	1	0

Category Breakdown

Suggestion	3
Documentation	0
Bug	7
Optimization	1
Good Code Practices	0

Code Maturity Evaluation

Code Audit

Clip Finance Secure, Automated Yield Solutions

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 as well as takes the correct measures in rounding the results of arithmetic operations.
Centralization	Weak. The owner and/or moderator have significant privileges over the funds.
Code Stability	Moderate. Some functionality was being added throughout the audit.
Upgradeability	Satisfactory. UUPS proxies are used in most smart contracts.
Function Composition	Satisfactory. Functionality was well split into functions.
Front-Running	Satisfactory. No significant frontrunning opportunities were found.
Monitoring	Satisfactory. Events are correctly emitted for the most significant state changes.
Specification	Satisfactory. The code matched the design specifications.
Testing and Verification	Moderate. Unit tests were present for most functionality but fuzzing and invariant tests could be performed.

Findings

Code Audit

Clip Finance Secure, Automated Yield Solutions

Findings

3S-Clip-H01

In **`PCLBaseSwapInside::_compoundRewards()`**, if the **`sellPortion`** is smaller than **`rewardSwapThreshold`**, some of the rewards will move to **`keepPortion`**

Id	3S-Clip-H01
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Addressed in #2464036.

Description

Rewards allocated to compounding shares are only swapped to tokens 0 and 1 if the **`sellPortion`** exceeds the **`rewardSwapThreshold`**. However, the allocated sell portion is not accounted for anywhere, which means that if the rewards are not sold due to being smaller than the threshold, a portion of them will go to the **`keepPortion`** the next time **`_compoundRewards()`** is called.

In the worst scenario, sell portion may always be smaller than the threshold if **`_compoundRewards()`** is called frequently enough and no rewards will be allocated to compounding shares.

Recommendation

Add a variable tracking the sell portion to make sure rewards never transition to the keep portion.

3S-Clip-H02

`PHyperLPoolSwapInside::_rebalance()` checks the price manipulation before and after the swap with the same price threshold

Id	3S-Clip-H02
Classification	High
Severity	High
Likelihood	Medium
Category	Bug
Status	Addressed in #0c8de23.

Description

PHyperLPoolSwapInside::_rebalance() checks price manipulation before swapping to prevent sandwich attacks and after swapping as a way to mitigate slippage. The same **`priceThreshold`** is used for both checks, which means that when rebalancing a large amount in comparison to the pool's liquidity, which would lead to significant slippage, an equally large **`priceThreshold`** would need to be used. This means that the **`PHyperLPoolSwapInside`** would be vulnerable to sandwich attacks.

Recommendation

Use 2 different **`priceTreshold`** variables, the first one smaller to guarantee that no sandwich attacks will happen and a second one, which could be bigger, to check slippage.

3S-Clip-M01

Compounding shares will overtime get less rewards than non compounding shares

Id	3S-Clip-M01
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Status	Addressed in #2464036.

Description

`keepPortion` and **`sellPortion`** are based on the total number of shares of each type. However, **`compoundShareRate`** is expected to increase due to accumulating rewards. This means that, for compounding shares, the same liquidity minted compared to non compounding shares will lead to less shares, when depositing. Thus, over time, assuming equal deposits of compounding and non compounding shares, the total supply of non compounding shares is expected to grow more than compounding shares.

Recommendation

`keepPortion` and **`sellPortion`** should be calculated based on the total liquidity of each share type, not the total supply, as liquidity is effectively what is generating rewards.

3S-Clip-L01

InRatioSwap fuzz test shows error up to **1e10** of a token with 18 decimal places

Id	3S-Clip-L01
Classification	Low
Severity	Low
Likelihood	High
Category	Bug
Status	Addressed in #41d3c39.

Description

Performed fuzz tests of the **InRatioSwap.getInRatioSwap()** function showing that the error is always smaller than **1e10**.

Here is the test for reference:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console} from "forge-std/Test.sol";
import {IERC20Metadata} from
"contracts/deps/OpenZeppelinV5/IERC20Metadata.sol";
import {SafeERC20} from "contracts/deps/OpenZeppelinV5/SafeERC20.sol";
import {IPancakeV3Pool} from
"contracts/interfaces/pancake/IPancakeV3Pool.sol";
import "contracts/interfaces/pancake/IPancakeV3PositionManager.sol";
import "contracts/lib/pancake/InRatioSwap.sol";

contract InRatioSwapTest is Test {
    using SafeERC20 for IERC20Metadata;
    address token0 = 0x55d398326f99059fF775485246999027B3197955 ;
    address token1 = 0xc5f0f7b66764F6ec8C8Dff7BA683102295E16409;
    V3PancakePool pool =
V3PancakePool.wrap(0xbF72B6485E4b31601aFe7B0a1210Be2004D2B1d6);
    IPancakeV3PositionManager positionManager =
IPancakeV3PositionManager(0x46A15B0b27311cedF172AB29E4f4766fbE7F4364);
    int24 currentTick;
```

```
    uint24 fee = 100;
    function setUp() public {
        vm.createSelectFork("https://bsc.meowrpc.com", 37430470);
        (, currentTick) = pool.sqrtPriceX96AndTick();
        IERC20Metadata(token0).approve(address(positionManager),
type(uint256).max);
        IERC20Metadata(token1).approve(address(positionManager),
type(uint256).max);
    }
    function testFuzz(uint256 token0Amount, uint256 token1Amount, int24
tickLower, int24 tickUpper) public {
        vm.assume(tickLower < tickUpper);
        vm.assume(tickLower > TickMath.MIN_TICK);
        vm.assume(tickUpper < TickMath.MAX_TICK);
        token0Amount = token0Amount %
IERC20Metadata(token0).balanceOf(V3PancakePool.unwrap(pool));
        token1Amount = token1Amount %
IERC20Metadata(token1).balanceOf(V3PancakePool.unwrap(pool));
        deal(token0, address(this), token0Amount);
        deal(token1, address(this), token1Amount);
        (uint256 amountIn,, bool zeroForOne,) =
InRatioSwap.getInRatioSwap(pool, tickLower, tickUpper, token0Amount,
token1Amount);

        uint160 swapThresholdPrice = zeroForOne ? TickMath.MIN_SQRT_RATIO +
1 : TickMath.MAX_SQRT_RATIO - 1;
        if (amountIn != 0)
            IPancakeV3Pool(V3PancakePool.unwrap(pool)).swap(address(this),
zeroForOne, int256(amountIn), swapThresholdPrice, "");
        uint256 amount0ToMint =
IERC20Metadata(token0).balanceOf(address(this));
        uint256 amount1ToMint =
IERC20Metadata(token1).balanceOf(address(this));
        try
            positionManager.mint(
                    IPancakeV3PositionManager.MintParams({
                        token0: token0,
                        token1: token1,
                        fee: fee,
                        tickLower: tickLower,
                        tickUpper: tickUpper,
                        amount0Desired: amount0ToMint,
```

```
                        amount1Desired: amount1ToMint,
                        amount0Min: 0,
                        amount1Min: 0,
                        recipient: address(this),
                        deadline: block.timestamp
                    })
            )
        {
            assertLt(IERC20Metadata(token0).balanceOf(address(this)), 1e10);
            assertLt(IERC20Metadata(token1).balanceOf(address(this)), 1e10);
        }
        catch (bytes memory reason) {
            require(amount0ToMint == 0 || amount1ToMint == 0, "oops");
        }
    }
    function pancakeV3SwapCallback(int256 amount0Delta, int256 amount1Delta,
bytes calldata /*data*/) external {
        if (amount0Delta > 0)
IERC20Metadata(token0).safeTransfer(msg.sender, uint256(amount0Delta));
        else if (amount1Delta > 0)
IERC20Metadata(token1).safeTransfer(msg.sender, uint256(amount1Delta));
    }
}
```

Recommendation

The **`_deposit()`** function should return the unused tokens to the **`msg.sender`**.

3S-Clip-L02

Missing **`_disableInitializers()`** call in the constructor

Id	3S-Clip-L02
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Addressed in #0c8de23.

Description

`PCLBaseSwapInside()` is missing a **`_disableInitializers()`** call in the constructor, allowing the implementation contract to be initialized. This is not a security concern at the moment but could be in a future version of the implementation.

Recommendation

Add **`_disableInitializers()`** to the constructor.

3S-Clip-L03

Rebalancing back to original **`lower`** and **`upper`** ticks will not deposit **`nft`** to farm

Id	3S-Clip-L03
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Addressed in #0c8de23.

Description

Rebalance changes the **`lower`** and **`upper`** ticks in **`PHyperLPoolSwapInside`** and withdraw from the farming contract if it is allocated. If the rebalance is performed to a previously used **`lower`** and **`upper`** ticks, the **`nftId`** will not be 0 and it will add liquidity instead of minting. As can be seen, the add liquidity branch does not deposit to the farming contract if it is not allocated yet.

Recommendation

The impact is reduced as it can be deposited later by calling PHyperLPoolSwapInside::depositNftToFarm(). Consider try allocating after the **`_mint()`** call if it is not allocated yet (and setting the allocatedToFarming bool to true)

3S-Clip-L04

`PHyperLPoolSwapInside::_swapWithCustomData()` `.approve()` reverts for tokens that don't return a boolean

Id	3S-Clip-L04
Classification	Low
Severity	Medium
Likelihood	Low
Category	Bug
Status	Addressed in #0c8de23.

Description

`PHyperLPoolSwapInside::_swapWithCustomData()` uses unsafe **`.approve()`** which does not return a bool on some tokens, reverting.

Recommendation

Use **`.forceApprove()`**.

3S-Clip-L05

`PHyperLPoolSwapInside::_swapWithCustomData()` checks price manipulation for **`pool`**, but the **`router`** used may route through another pool

Id	3S-Clip-L05
Classification	Low
Severity	Medium
Likelihood	Low
Category	Suggestion
Status	Acknowledged

Description

PHyperLPoolSwapInside::_swapWithCustomData() checks price manipulation for **`pool`**; however, the **`router`** chosen may route through a different pool, rendering the price manipulation useless and leading to MEV.

Recommendation

Either enforce that the router used routes through the pool or do the price manipulation check for the pool that the router routed through.

3S-Clip-L06

`PHyperLPoolSwapInside::_swapWithCustomData()` should check if the router has code

Id	3S-Clip-L06
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Addressed in #0c8de23.

Description

PHyperLPoolSwapInside::_swapWithCustomData() performs swap using a router provided as argument in **`swapData`**. Solidity performs contract length checks when using an interface, but not when using **`.call()`**, making this call dangerous as **`success`** will be true.

Recommendation

Use Openzeppelin's Address::functionCall() which performs all the checks.

3S-Clip-L07

Lost NFT if **`farmingContract`** is incorrectly set

Id	3S-Clip-L07
Classification	Low
Severity	Low
Likelihood	Low
Category	Suggestion
Status	Addressed in #0c8de23.

Description

In PHyperLPoolSwapInside::_depositNftToFarm(), the NFT transfer will succeed if the farmingContract is incorrectly set. This is a possibility as the contracts are intended to be deployed in more than 1 chain, with different addresses.

Recomendation

It should be enough checking if the address has code in the constructor **`if (farmingContract.code.length != 0) revert()`**.

3S-Clip-N01

Unnecessary `abi.decode` in `PHyperLPoolSwapInside::_swapWithCustomData()`

Id	3S-Clip-N01
Classification	None
Category	Optimization
Status	Addressed in #0c8de23.

Description

PHyperLPoolSwapInside::_swapWithCustomData() does `abi.decode` of the `returnData`, which is already a bytes variable.

Recommendation

`revert(string(returnData));` should be enough