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 audited Clip Finance in a 4 person week engagement. The audit was conducted from 20/11/2023 to 1/12/2023. It focused on the core smart contracts, responsible for managing liquidity and allocating to strategies.

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

```
├── admin.sol
├── BatchOut.sol
├── Batch.sol
├── create2
│   ├── Create2Deployer.sol
│   └── PlaceholderContract.sol
├── exchange
│   ├── IzumiSwapPlugin.sol
│   ├── PancakeSwapPlugin.sol
├── idle-strategies
│   └── DefaultIdleStrategy.sol
├── oracles
│   ├── ChainlinkOracle.sol
├── ReceiptNFT.sol
├── Registry.sol
├── Rewards.sol
├── SharesToken.sol
├── StrategyRouterLib.sol
├── StrategyRouter.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	5e5452a
Language	Solidity
Platform	Linea

Engagement Summary

Timeline	20/11/2023 to 1/12/2023
№ of Auditors	2
Review Time	4 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	7	6	1
Medium	8	6	2
Low	2	0	2
None	0	0	0

Category Breakdown

Suggestion	0
Documentation	0
Bug	17
Optimization	0
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	Weak. Functionality was being added throughout the audit.
Upgradeability	Satisfactory. UUPS proxies are used in most smart contracts.
Function Composition	Moderate. Some functions such as rebalance could be split in internal functions to reduce complexity.
Front-Running	Moderate. Some front-running issues were addressed but there are still ways to abuse it.
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 Finance-H01

Anyone can grief users, stopping them from fulfilling their withdrawals

Id	3S-Clip Finance-H01
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Addressed in #04bd247

Description

On the Batch.sol contract, anyone can call withdrawFulfill() with the current cycle id, incrementing the **`cycleInfo[currentCycleID].withdrawRequestsFullFilled`** and setting the **`userWithdrawStorage[cycleID][userAddress].withdrawStatus`** to true, without sending any tokens to the users, since the **`currentCycle.tokensWithdrawn.token`** will have a length of zero, so the loop will constantly skip iterations at this continue.

When someone inevitably calls **`executeBatchWithdrawFromStrategyWithSwap()`** and then **`withdrawFulfill()`**, the loop will start with the offset **`cycleInfo[currentCycleID].withdrawRequestsFullFilled`** and some users will have the **`userWithdrawStorage[cycleID][userAddress].withdrawStatus`** flag set to true, so they will be skipped and will never be able to receive the tokens from their shares.

This issue is also problematic because, if later but still in the same cycle, the same users try to add shares, they will be added to the previous request (which will always be skipped by the **`withdrawFulfill()`**), so they will also lose these extra shares.

Recommendation

Add **`require(cycleID < currentCycleId);`** at the start of **`withdrawFulfill()`**.

3S-Clip Finance-H02

Swapping with deadline as **`block.timestamp`** and 0 minimum amount out is vulnerable to MEV

Id	3S-Clip Finance-H02
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Addressed in #3c1437c

Description

`Exchange:swap()` sets **`minAmountOut`** to 0 and in the plugins the deadline is set to block.timestamp.

This means that miners may hold the transaction and do whatever MEV strategy they wish, stealing users who incur massive slippage.

Recommendation

Send block.timestamp as argument to the functions that call swap and forward it. Additionally, **`Exchange:calculateMinAmountOut()`** should always be used to protect from MEV, even if not for stablecoins.

3S-Clip Finance-H03

`BatchOut:executeBatchWithdrawFromStrategyWithSwap()` gives unfairly different slippage depending on the chosen token

Id	3S-Clip Finance-H03
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Acknowledged

Description

`BatchOut:executeBatchWithdrawFromStrategyWithSwap()` calls `strategyRouter:withdrawFromStrategies()` with the token having the most requests shares. This call will keep the assets/shares ratio, as it burns a number of shares according to the assets in `strategyRouter:calculateSharesUsdValue()`. However, the tokens actually withdrawn could be significantly less than the ideal corresponding to the shares.

After withdrawing, it loops through the other tokens and calculates the ideal usd value from their requested shares. This means that only the token with the max shares that was used to withdraw is taking the slippage, while the others are still getting their maximum usd value (slightly less as they will be swapped, incurring extra slippage).

Thus, users that chose to withdraw in the most requested token will likely incur significantly more slippage than the other tokens. The opposite scenario might also happen, if the `withdrawToken` has enough balance without going through swaps in `router.withdrawFromStrategies()`, then the slippage is taken only on the other tokens.

Recommendation

Consider creating cycles only of a certain withdrawal token, so users are under the same slippage. Else, a function could be created that handles withdrawals in several tokens, but this would take a much bigger effort.

3S-Clip Finance-H04

`BatchOut:withdrawFulfill()` can be DoSed by spamming withdrawal requests, leading to OOG reverts

Id	3S-Clip Finance-H04
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Addressed in #4dc4687

Description

Anyone can schedule as many withdrawals as they want in **`BatchOut:scheduleWithdraw()`**, specifying little shares to different **`withdrawTo`** addresses. Then, in **`withdrawFulfill()`**, it will loop over all the requests in the cycle, leading to OOG reverts if enough requests were spammed.

Recommendation

Send as argument a number of requests to fulfill, so the OOG revert can be prevented.

3S-Clip Finance-H05

Scheduled withdrawals with unsupported tokens will be halted

Id	3S-Clip Finance-H05
Classification	High
Severity	High
Likelihood	Medium
Category	Bug
Status	Addressed in #8f3f6e4

Description

A token might be supported at time A, letting users withdraw in **`BatchOut`** using this token. However, if the token is not supported anymore after withdrawals are scheduled, it won't be possible to call **`BatchOut:executeBatchWithdrawFromStrategyWithSwap()`**, as it will revert when trying to call **`router.withdrawFromStrategies()`**, here.

The workaround is supporting the token for a short period of time again to let users fulfill their withdrawals, but this could lead to other problems, given that it was previously deprecated.

Recommendation

Change the withdraw token to another supported token in **`executeBatchWithdrawFromStrategyWithSwap()`** if the token is no longer supported.

3S-Clip Finance-H06

Halted withdrawals in **`BatchOut`** due to setting **`withdrawTo`** to **`address(0)`** in **`scheduleWithdrawal()`**

Id	3S-Clip Finance-H06
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Addressed in #4dc4687

Description

`BatchOut:scheduleWithdraw()` allows sending a **`withdrawTo`** argument of 0. Some tokens, such as USDT, revert when transferring tokens to address 0. This means that, when fulfilling withdrawals, the loop will be DoSed when it reaches the 0 address withdrawal, halting all withdrawals in the cycle.

Recommendation

Do not allow 0 address **`withdrawTo`**.

3S-Clip Finance-H07

Yield loss due to **`StrategyRouterLib:rebalanceStrategies()`** not allocating saturated strategy deposits

Id	3S-Clip Finance-H07
Classification	High
Severity	High
Likelihood	High
Category	Bug
Status	Addressed in #ed8e3c9

Description

`StrategyRouterLib:rebalanceStrategies()` withdraws excess funds from strategies and adds it up to the router balance. When strategies have less funds than desired, it first stores in strategyDatas[i].toDeposit the amount to be deposited. As can be seen in the previous code link, the strategy was not completely saturated, so the weight is bigger than 0, the funds were sent to the strategy, but **`deposit()`** was not called yet.

Then, if the remaining weight of the strategy is less than ALLOCATION_THRESHOLD, it skips the current strategy, never depositing the previous saturated weight.

If it is bigger than **`ALLOCATION_THRESHOLD`**, it still skips the deposit of the previously saturated weight, due to the fact that it overrides **`strategyDatas[i].toDeposit`** (L639, L662 and L684).

Thus, when it calls `strategy.deposit()`, it only sends the value of the last swap, without taking into account the previously accumulated **`strategyDatas[i].toDeposit`** values.

Recommendation

Change **`strategyDatas[i].toDeposit = received;`** to **`strategyDatas[i].toDeposit += received;`** in L639, L662 and L684.

Also, when the remaining weight of the strategy equals less than `ALLOCATION_THRESHOLD`, deposit **`strategyDatas[i].toDeposit`** before continuing.

3S-Clip Finance-M01

Potential overflow in **`PancakeSwapPlugin:getRoutePrice()`**

Id	3S-Clip Finance-M01
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in #3d2ad8c

Description

Currently the price is calculated as **`uint256 routePrice = FullMath.mulDiv(uint256(sqrtPriceX96) * uint256(sqrtPriceX96), precision0, 2 ** (96 * 2));`**. Doing **`uint256(sqrtPriceX96) * uint256(sqrtPriceX96)`** might overflow as 2 uint160 variables may result in a number with more than 256 bits. The correct way of calculating this is in **`OracleLibrary:getQuoteAtTick()`**, here.

Note: also here.

3S-Clip Finance-M02

DoSed `StrategyRouter:withdrawFromStrategies()` if `strategyTokenBalancesUsd[i]` is too small in the swapping phase

Id	3S-Clip Finance-M02
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Addressed in #687172d

Description

`StrategyRouter:withdrawFromStrategies()` withdraws from the idle strategy of the withdraw token and then tries to withdraw from idle strategies and later strategies with other tokens.

It tries to swap from idle strategies and strategies if there is still not enough usd, skipping the swap if idleStrategyTokenBalancesUsd or strategyTokenBalancesUsd are 0. However, when the balance is very small, it will likely be swapped into a 0 amount due to slippage, making it revert.

This can be exploited by griefers by sending 1 amount to an idle strategy so it reverts.

Recommendation

Define a threshold to swap, similarly to the allocation threshold.

3S-Clip Finance-M03

Halted withdrawals in **`BatchOut:withdrawFulfill()`** due to tokens **`transfer()`** reverting on 0 transfer amount

Id	3S-Clip Finance-M03
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug
Status	Addressed in #48a6f08

Description

Some tokens may revert on 0 amount transfers. This means that if someone schedules a withdrawal of 1 share, it could convert to a 0 amount and halt all the other withdrawals.

Recommendation

Skip the transfer if the amount to transfer is 0.

3S-Clip Finance-M04

`Batch:withdraw()` can be DoSed by frontrunning it with `strategyRouter:allocateToStrategies()`

Id	3S-Clip Finance-M04
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Acknowledged

Description

Anyone who wants to withdraw from a batch via **`Batch:withdraw()`** can be frontrunned by an **`allocateToStrategies()`** call, which increases the cycle id, making the withdrawal revert. This not only DoS users, but also places them at a loss if they want to withdraw, as they will likely receive shares worth less than their deposits at the moment the allocation happens.

Recommendation

Make **`strategyRouter:allocateToStrategies()`** permissioned so it becomes impossible for malicious users to perform the attack. Additionally, it's probably better to set some sort of fixed interval for the allocations to happen, so users have time to withdraw if they want to.

3S-Clip Finance-M05

Inconsistent **`batch:rebalance()`** behavior when some strategies reach their limit, leading to yield loss

Id	3S-Clip Finance-M05
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug
Status	Addressed in #687172d

Description

Strategies are allocated funds pro-rata to their weights. However, when the target of a strategy is reached, it won't be allocated any more funds.

The funds that were supposed to be allocated to a strategy but its limit was reached, will be allocated to other strategies if they are after it in the array https://github.com/ClipFinance/strategy-router/blob/master/contracts/Batch.sol#L335.

However, if the strategies that haven't reached their limit are before the strategy whose limit was reached in the array, the extra funds will be allocated to idle strategies instead https://github.com/ClipFinance/strategy-router/blob/master/contracts/Batch.sol#L288.

This is because when a strategy reaches its limit, the weight is set to 0, and the total batch unallocated tokens are not decreased. This means that if the strategy is first, the other strategies will acquire the extra unallocated tokens. But if the strategy is last, the tokens won't be allocated and will be sent to idle strategies instead.

Recommendation

Cap the weight of a strategy to the amount required for the strategy to reach its limit, before allocating any funds.

Note: **`rebalanceStrategies()`** ignores the capacity data, which means that it would likely surpass the limit if called afterwards.

3S-Clip Finance-M06

StrategyRouterLib.sol bug when subtracting from balance

Id	3S-Clip Finance-M06
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Status	Addressed in #ed8e3c9

Description

Lines 665 and 666 of the StrategyRouterLib have the following code:

```
currentTokenDatas[j].currentBalance -= desiredAllocationUniform;
currentTokenDatas[j].currentBalanceUniform -= desiredAllocationUniform; //
not actually in uniform format
```

Here, variable **`desiredAllocationUniform`** actually holds the original desired allocation value (not in the uniform format), so in line 666 the subtraction is being performed by variables in different formats. Depending on the difference between the original and the uniform decimals, one of the three scenarios should happen:

- if the number of decimals in the same, the code should run without a problem

- if the number of original decimals is higher than the number of uniform decimals, the execution will most likely underflow and revert in line 666.

- if the number of original decimals is smaller than the number of uniform decimals, the contract will try to transfer tokens it does not have and revert.

- no under/overflow happens but the accounting becomes incorrect.

Recommendation

Change line 666 to **`currentTokenDatas[j].currentBalanceUniform -= toUniform(desiredAllocationUniform, supportedTokensWithPrices[j].token);`**

3S-Clip Finance-M07

Fee on transfer tokens transfer less tokens than what is stored in the receipt on deposits

Id	3S-Clip Finance-M07
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Bug
Status	Acknowledged

Description

Some tokens have a fee on transfer, such as USDT, although it is disabled currently (line 127 and 177).

This means that on deposits, the actually deposited amount to the smart contract will not be the one passed as argument, but its value minus the fee. Thus, for example, the `Batch.sol` contract will hold less funds than stored, potentially leading to withdrawal failure.

Recommendation

The actual transferred amount is the balance after minus the balance before the transfer. Modify this line to:

```
uint256 previousBalance = IERC20(depositToken).balanceOf(address(batch));
IERC20(depositToken).safeTransferFrom(msg.sender, address(batch),
depositAmount);
depositAmount = IERC20(depositToken).balanceOf(address(batch)) -
previousBalance;
```

Also, place the lines above before calling **`batch.deposit()`**.

3S-Clip Finance-M08

Tokens with callbacks may allow malicious attackers to steal the protocol

Id	3S-Clip Finance-M08
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug
Status	Addressed in #972526f

Description

In **`Batch.sol`**, function **`withdraw()`** burns the **`receiptId`** after transferring the tokens to the user. This means that if the token has a callback, the user may use it to sell the nft somewhere else, getting some extra value for it, and then the execution continues in **`withdraw()`**, burning the nft of the other new owner.

In **`BatchOut.sol`**, **`withdrawFulfill()`**, the **`withdrawStatus`** is set to true only after the tokens are transferred to the user. An attacker could use a callback to reenter the function and withdraw the shares until the contract is drained.

Recommendation

Follow the checks-effects-interactions pattern.

In **`Batch.sol`**, burn the receipt and only then transfer the tokens to the **`receiptOwner`**.

In **`BatchOut.sol`**, set the **`withdrawStatus`** to true before transferring tokens.

3S-Clip Finance-L01

Missing fee refund on **`Batch.sol`**

Id	3S-Clip Finance-L01
Classification	Low
Category	Bug
Status	Acknowledged

Description

The **`msg.value`** sent to **`Batch:deposit()`** might be bigger than the **`depositFeeAmount`**, which means that the contract will receive more native than the fee.

Recommendation

Place a refund mechanism which gives back to the depositor the excess BNB.

3S-Clip Finance-L02

`getDepositFeeInBNB()` assumes a stablecoin price of 1 USD, which may not be true if it depegs

Id	3S-Clip Finance-L02
Classification	Low
Category	Bug
Status	Acknowledged

Description

In **`Batch.sol`**, function `getDepositFeeInBNB()` calculates the fee amount in BNB, considering that the stablecoin price is 1 USD, which may not be true, as we've seen with USDC in the past.

It also means that if a coin other than a stablecoin is used to deposit, a much smaller/larger fee would be triggered.

For example, 20 tokens are deposited, the fee is 10%, so 2 fee amount.

The token price is 10 USD.

The bnb price is 250 USD.

The fee amount in BNB would be 2/250 = 0.008.

So the user would deposit 20 tokens, worth 200 USD and pay a fee of 0.008 BNB or 2 USD, which is 2/200 = 1%, instead of 10%.

The real fee, according to the code, is depositFee/(token price in USD).

Recommendation

Consider fetching the price of BNB in units of the to be deposited token.

This way, the fee in the code will match the real percentage in USD.