BAIL security

BAILSEC.IO

EMAIL : OFFICE@BAILSEC.IO

TWITTER : @BAILSECURITY

TELEGRAM : @HELLOATBAILSEC

FINAL REPORT:

Cairo Finance

15 May 2023

Disclaimer:

Security assessments are limited by time and the information provided by the client. Therefore, the findings in this report should not be considered a complete list of security issues, flaws, or defects in the target system or codebase.

The content of this assessment is not an investment. The information provided in this report is for general informational purposes only and is not intended as investment, legal, financial, regulatory, or tax advice. The report is based on a limited review of the materials and documentation provided at the time of the audit, and the audit results may not be complete or identify all possible vulnerabilities or issues. The audit is provided on an "as-is," "where-is," and "as-available" basis, and the use of blockchain technology is subject to unknown risks and flaws.

The audit does not constitute an endorsement of any particular project or team, and we make no warranties, expressed or implied, regarding the accuracy, reliability, completeness, or availability of the report, its content, or any associated services or products. We disclaim all warranties, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We assume no responsibility for any product or service advertised or offered by a third party through the report, any open-source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, and the related services and products. We will not be liable for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract.

The contract owner is responsible for making their own decisions based on the audit report and should seek additional professional advice if needed. The audit firm or individual assumes no liability for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract. The contract owner agrees to indemnify and hold harmless the audit firm or individual from any and all claims, damages, expenses, or liabilities arising from the use or reliance on the audit report or the smart contract.

By engaging in a smart contract audit, the contract owner acknowledges and agrees to the terms of this disclaimer.

1.Executive Overview

This audit report has been prepared for Cairo Finance regarding its operations on the blockchain. The purpose of this report is to provide an independent assessment of Cairo Finance's adherence to secure coding practices and reliability of its operations on the blockchain.

Cairo Finance is a new financial institution leveraging the capabilities of the blockchain to offer decentralized financial services. The utilization of blockchain technology presents unique opportunities and challenges, particularly in terms of security, transparency, and auditability. As a result, it is crucial to evaluate Cairo Finance's adherence to industry standards and to identify any potential vulnerabilities or risks within its blockchain-based operations.

This audit report has been conducted by an experienced team of auditors who specialize in blockchain technology and have a comprehensive understanding of the blockchain ecosystem. The audit process involved a thorough review of Cairo Finance's smart contracts, transactional data, security measures, and compliance protocols. Additionally, the auditors engaged in extensive discussions with Cairo Finance's management and technical teams to gain insights into their operational procedures and risk management practices.

The scope of this audit report encompasses an evaluation of Cairo Finance's smart contracts, including their design, implementation, and execution. It also includes an assessment of the overall security posture of the platform, focusing on the identification and mitigation of potential vulnerabilities, such as code exploits, unauthorized access, and fraudulent activities.

Throughout this report, we will present our findings, recommendations, and observations in a structured manner, providing Cairo Finance with actionable insights to enhance its operational efficiency, security, and compliance with industry standards. It is important to note that this audit report represents an independent assessment of Cairo Finance's operations on the blockchain, based on the information available at the time of the audit.

It is recommended that the management of Cairo Finance carefully review the contents of this report, and take appropriate actions to address the identified issues and implement the recommended improvements. By doing so, Cairo Finance can further strengthen its position as a trusted and secure provider of decentralized financial services on the blockchain.
Please note that this report is confidential and intended solely for the use of Cairo Finance and its authorized personnel.

1. Project Details

Project	Cairo Finance
Website	https://cairofinance.app/
Type	Decentralized Financial Services (DEFI)
Language	Solidity
Methods	Manual review
Deployed contract	Not deployed

2. Detections Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	13			
Medium	9			
Low	5			
Informational	12			
Total	39			

2.1 Detections Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users

3. Global Issues

Issue	Unclean contract architecture
Severity	Informational
Description	The contract exhibits the presence of multiple base contracts that inherit external libraries. To illustrate, the main contract, named "Cairo," inherits both the "BaseConfig" contract and the "StableCoinStrategy" contract. In turn, the "StableCoinStrategy" contract inherits the "BaseConfig" contract, while both contracts inherit the "Initializable" library. Notably, the SafeERC20 is configured redundantly. This behavior is deemed unnecessary, leading to the presence of cumbersome code and the potential for confusion among external reviewers.
Recommendations	Our recommendation entails conducting a thorough cleanup of the contract architecture, specifically addressing the elimination of redundant and unnecessary inheritances. It is advised to remove any duplicated inheritances and ensure that all dependencies are initialized correctly, with the intention of streamlining the codebase. This course of action aims to enhance the overall efficiency and maintainability of the contract, reducing complexity and potential confusion for future reviewers and stakeholders.
Comments	

4. BaseConfig

The **BaseConfig** contract serves as an abstract contract that is inherited by the Cairo contract.

It provides access to essential base variables such as the reward token, important roles, lp token, staking contract, and other pertinent elements. In the process of initialization, all the required variables are assigned and necessary approvals are granted.

Issue	Missing Validation
Severity	Low
Description	The __BaseConfig_init function initializes important variables that should be under no circumstances address(0)
Recommendations	We recommend addressing the issue related to the __BaseConfig_init function. The function should be modified to ensure that important variables are never initialized with the address(0) value.
Comments	

Issue	Unused variables
Severity	Informational
Description	The contract contains variables that are unused, such as FUNDS_RECOVERY_ROLE, ZAP_ROLE, MAX_PERCENTAGE
Recommendations	To address the issue of unused variables in the contract, it is recommended to perform the following steps: 1. Review the purpose and intended functionality of the variables FUNDS_RECOVERY_ROLE, ZAP_ROLE, and MAX_PERCENTAGE in the contract code.

	2. If these variables are indeed unnecessary or not being utilized, remove them from the contract to improve code clarity and efficiency. 3. Ensure that the removal of these variables does not have any unintended side effects on the overall functionality of the contract. 4. Document the rationale behind removing these variables in the contract's code comments or accompanying documentation to provide clarity to future developers and auditors.
Comments	

Issue	Unnecessary CAFToken approval
Severity	Informational
Description	The referral contract does not require approval for the CAFToken.
Recommendations	We recommend removing the approval requirement for the CAFToken. This will simplify the contract's logic and reduce the complexity of token interactions.
Comments	

Issue	Paused checks are redundant
Severity	Informational
Description	Since the PausableUpgradeable library already exposes whenPaused and whenNotPaused modifiers, the isNotPaused and isPaused functions can be removed.
Recommendations	Remove these functions.
Comments	

5. StableCoinStrategy

The **StableCoinStrategy** contract serves as an abstract contract that is inherited by the Cairo contract. Its primary responsibility lies in the accounting logic pertaining to deposits and withdrawals This contract is a fork from Grizzly Finance.

When users deposit tokens, the **stableCoinStrategyDeposit** function is invoked, which takes into consideration the **rewardMask**, total deposited amount, and total reinvested amount. Conversely, withdrawals are managed through the **stableCoinStrategyWithdraw** function.

Prior to each regular deposit, the **Cairo** contract handles reward compounding and subsequently calls the **stableCoinStrategyUpdateRewards** function to augment the roundMask.

It is important to note that the contract employs EIP 1973 as the underlying reward distribution logic.

Issue	Users can make other user's compounded amount work for themselves
Severity	Medium

Description	The highly customized reward logic employed in the system introduces risks. Although the logic has been utilized by Grizzly Finance for a considerable period, it is essential to acknowledge that it may not be flawless. Our analysis has identified a flaw within the reward logic that allows users to exploit other users' inactivity for personal gains through compounding.
	
	The proof of concept (PoC) scenario is as follows:
	
	1. Alice initially deposits 10,000 LP tokens:
	
	roundMask = 1
	
	participantData[Alice].rewardMask = 1;
	participantData[Alice].amount = 10_000;

participantData[Alice].totalReinvested = 0;
stablecoinStrategyDeposits += 0 - 0 + 10_000;

2. Over time, Alice's stake accumulates rewards, to make the example easy and clear let's say our stake is 10% of the total stake in PCS and the total reward per second, converted to LP tokens is **10 tokens/sec** which means that our vault receives 10 tokens/sec as reward (we ignore the whole zapping logic for now) and no other user deposits.

3. Bob now deposits 1 LP token after let's say **10_000** seconds, this means that Alice's reward of **100_000** LP token is now being compounded which essentially results in 110_001 LP tokens staked into the pancakeswap masterchef (110_000 from Alice, 1 from Bob).

The compounding also increases our percentage share in the PCS mc, resulting in a total share of over 50%, this means that our vault receives 55 LP tokens per second (converted) However, since Bob executed the deposit, the stableCoinStrategyDeposit variable is not increased by the compounded amount, as this would only happen if Alice deposits/harvests. This is the main flaw in the logic which allows Bob to misuse Alice's compound in order to receive more rewards.

roundMask = 10000000000001

participantData[Bob].rewardMask = 10000000000001;
participantData[Bob].amount = 1;
participantData[Bob].totalReinvested = 0;
stablecoinStrategyDeposits = 10_001;

4. However, Alice has not yet called deposit/withdraw which means that the stableCoinStrategyDeposit variable is still

sitting at 10_001 , as one can recognize in the example before.

5. After **1_000** seconds, bob harvests via withdraw(0), which results in **55_000** LP tokens for the vault.

The roundMask is now increased as follows:

roundMask += (1e12 * 55000) / 10001;

roundMask = 15499450054995

Bobs current Balance:
1 + ((15499450054995 - 10000000000001)* 1) / 1e12
= 6.499 = 6 LP tokens

participantData[Bob].rewardMask = 15499450054995;
participantData[Bob].amount = 6;
participantData[Bob].totalReinvested = 5;
stablecoinStrategyDeposits = 10_006;

The effective reward distribution during these **1_000** seconds were **55_000** LP tokens in total, the effective staked amount from Bob was 1 token, from Alice was 110_000, therefore Bob has 0.0009% staked compared to Alice which would result in 0.50 (rounded down would be zero) LP tokens, however, due to the flaw Bob received 5 LP tokens (rounded down)

6. Bob now profits from Alice's inactivity because her rewards are already compounded into the pancakeswap masterchef but the roundMask variable is not properly accounted for her compounded amount, resulting in way higher rewards for Bob than desired.

7. If Alice now executes a withdrawal, deposit or getUpdatedState immediately after Bob, without the flaw her

	balance would be almost all 110_000 + 55_000 tokens. Since the withdrawal is immediately after Bob's, the roundMask will not get increased. roundMask = 15499450054995 Now the calculated balance is fetched: balance = getStablecoinStrategyBalance(); resulting in the following calculation: return participantData[msg.sender].amount + ((roundMask - participantData[msg.sender].rewardMask) * participantData[msg.sender].amount) / DECIMAL_OFFSET; } 10_000 + ((15499450054995 - 1) * 10_000) / 1e12; = 164994 Therefore Alice lost 5 LP tokens due to her inactivity. This example can be amplified the longer Alice waits with updating her state.
Recommendations	In order to address the issue at hand, it is recommended to implement a solution that involves calculating the rewardMask in accordance with the accurate total staked amount. However, it is crucial to ensure that Bob's deposit is adjusted to reflect Alice's amount appropriately, as failure to do so would result in Alice continuing to incur losses in terms of rewards. This fix may prove to be complex, requiring thorough attention and consideration during the audit process.
Comments	

Issue	Minor issues
Severity	Informational
Description	The contract contains several minor issues, these will be aggregated under this section: This contract is abstract and is intended to be inherited by grizzly.sol. It is inherited by Cairo.sol
Recommendations	We recommended taking appropriate measures to rectify minor issues.
Comments	

6. Cairo

The main contract, known as "Cairo," serves as the primary interface for user interactions. It facilitates the deposit and withdrawal of the staking token, offering users the following three deposit functions:

1) The "**deposit**" function enables users to deposit BNB.
2) The "**depositLp**" function allows users to directly deposit the LP-Token.
3) The "**depositFromToken**" function enables users to deposit an arbitrary token.

Similarly, the contract provides flexibility in terms of withdrawal options:

1) The "**withdraw**" function permits users to withdraw as BNB.
2) The "**withdrawToToken**" function enables users to withdraw as an arbitrary token.

Additionally, the contract incorporates a referral system, wherein users can designate an address as their referrer. During a withdrawal, the designated referrer receives a 10% referral fee, applicable solely to the earned rewards. Similarly, the development team receives a 20% share, akin to the referral share, based on the earned rewards during each withdrawal.

A noteworthy feature is the unique "boost" functionality, which can be activated or deactivated by the UPDATER. This feature allows the distribution of the Cairo Finance token to stakers as a reward token. Although the reward distribution logic was derived from the grizzly finance protocol, the inclusion of the Cairo token reward sets it apart.

This contract is a modified fork of Grizzly Finance.

Issue	PAUSER can block withdrawals
Severity	Governance
Description	The PAUSER role can pause withdrawals, this can result in stuck funds.
Recommendations	Use a trusted multi-sig wallet as a pauser address.
Comments	

Issue	Wrongly configured DevTeam address can block withdrawals/compounds
Severity	Governance
Description	The aforementioned address receives fees during each compound and withdrawal. However, this address can be a smart contract without a fallback function resulting in a revert during the transfer of ether
Recommendations	Ensure that this address is a valid contract or EOA.
Comments	

Issue	Cairo reward addition is totally flawed
Severity	High
Description	The Cairo reward logic exhibits several issues, which are summarized as follows: **ONE:** The reward logic for the additional Cairo token follows a similar approach as the reward logic by Grizzly. However, both implementations utilize the same roundMask variable, albeit being private variables that do not conflict with each other. To ensure clarity and avoid potential confusion, it is advisable to rename them to indicate their distinct purposes. **TWO:** When a user triggers the withdraw function while having a stake below the rewardStakeThreshold, there is an omission of the updateRoundMask function call before the execution of _withdrawCafRewards. Furthermore, within the _withdrawCafRewards function, the updateRoundMask is invoked only after the user's reward calculation has already taken place. Subsequently, the assignment of participantMask[msg.sender] =

roundMask; follows immediately, resulting in the reset of the user's progress without disbursing the corresponding rewards.

THREE:
Flaw in RPT:
The calculation of the Reward Per Token (RPT) is as follows:
solidity
RPT = (boost_deposit_amount / (stablecoinStrategyDeposits / 1e18)) / 8760;

Assuming that RPT is equivalent to rewardPerToken, this computation does not yield the intended result, as rewardPerToken should essentially be boost_deposit_amount / stablecoinStrategyDeposits. However, to avoid rounding down, a precision multiplier is required. Consequently, this calculation may lead to potential side effects, including lower-than-expected rewards. One possible solution could involve adopting the masterchef logic. Additionally, the accumulation decreases with each withdrawal through the expressions:

boost_deposit_amount -= cafRew;
RPT = (boost_deposit_amount / (stablecoinStrategyDeposits / 1e18)) / 8760;

However, the roundMask continues to increase with each deposit. Consequently, in scenarios where significant stakers refrain from making withdrawals, the boost_deposit_amount fails to decrease (boost_deposit_amount -= cafRew;), while the roundMask can still be augmented by the undeducted amount (RPT = (boost_deposit_amount / (stablecoinStrategyDeposits / 1e18)) / 8760;). This situation results in an accumulation of rewards surpassing the total available rewards, potentially leading to a Denial-of-Service (DoS) condition. It is worth noting that this state can also arise from extended periods without any deposits or withdrawals, depending on the ratio of rewards to staked tokens.

FOUR:
Rewards are rounded down based on hourly intervals, as indicated by the following code snippet:

uint256 hoursPassed = (block.timestamp - lastStakeUpdated) / 1 hours;

Since rounding down always occurs, users can exploit this behavior by submitting a deposit every 59 minutes, thereby effectively preventing the distribution of any rewards. Nevertheless, this circumstance aligns with the regular business logic, where rewards are not distributed during such intervals.

FIVE:
Cairo rewards are computed based on the remaining amount after a withdrawal, as demonstrated by the following code excerpt:

uint256 user_deposit_amount = participantData[msg.sender].amount;
uint256 cafRew = (roundMask - userMask) * user_deposit_amount;

Regrettably, during a withdrawal, the value of participantData[msg.sender].amount is already subtracted before executing this calculation, resulting in a loss of rewards for the affected users.

SIX:
The updateRoundMask function employs stableCoinStrategyDeposits, which is related to the issue within stableCoinStrategy whereas users can profit from the inactivity of other users

SEVEN:
The act of toggling the boostStatus from false to true can lead to users receiving rewards retroactively. The following Proof of Concept

(PoC) outlines this behavior:

1. Initially, the boostDeposit function is invoked with a token quantity of 100.
2. Users begin accumulating and claiming rewards based on their participation.
3. At a certain point, the boostStatus is switched off.
4. A considerable amount of time elapses since users last updated their mask, let's assume 10,000 seconds.
5. Subsequently, the boostStatus is set to true once again.
6. Users will receive rewards retrospectively, considering the time interval since the last invocation of lastStakeUpdated, which is calculated as follows:

```
uint256 hoursPassed = (block.timestamp - lastStakeUpdated) / 1 hours;
```

Here, lastStakeUpdated represents a timestamp from 10,000 seconds ago.

EIGHT:

During a regular deposit, users will forfeit all Cairo rewards. The following Proof of Concept (PoC) illustrates this scenario:

1. Initially, a user has 100 tokens staked, with the current Cairo RM (Reward Mask) value set to 100.
2. The user proceeds to call the deposit function, which updates the user's RM to 100. Additionally, the deposit function modifies the RM to, let's say, 200.
3. Consequently, the user should receive tokens based on the difference between 200 and 100. However, no distribution of rewards occurs.
4. At the conclusion of the deposit function, the user's RM is set to 200 with the statement participantMask[msg.sender] = roundMask;.
5. As a result, the user effectively loses all accumulated rewards.

	Consider the following additional scenario: 1. Alice has 100 tokens staked, with the current Cairo RM set to 100. 2. Bob updates the Cairo RM, resulting in a new value of 200. 3. Subsequently, Alice calls the deposit function, which sets Alice's RM to 200. 4. As a consequence, Alice effectively loses all previously accumulated rewards.
Recommendations	Whenever a project incorporates custom reward logic, it is highly likely that it introduces flaws. This fact was once again demonstrated by the presence of an issue within the StableCoinStrategy contract, which persisted for over a year in the GrizzlyFi contract. There are two widely used reward systems for a reason: the masterchef logic and the synthetix staking rewards. Additionally, for top-up rewards, one could consider adopting the logic used in the gauges from solidly forks. At present, the Cairo reward logic is rendered inoperable. Even if the aforementioned issues are resolved, we maintain significant doubts regarding the effectiveness of this custom logic. Hence, we strongly recommend that the client transition to an alternative reward distribution logic, such as synthetix staking rewards or masterchef logic. Drawing upon our extensive expertise in auditing masterchefs, vaults, and staking contracts, we are 99% certain that merely addressing these issues and implementing the current custom logic will not suffice.
Comments	

Issue	Possible DoS in withdraw for users
Severity	**High**
Description	During the withdrawCafRewards function, the following section is implemented: if (participantData[msg.sender].amount == 0) { totalUsers - 1;} Unfortunately, the totalUsers variable was never increased at all which will result in an underflow in the aforementioned section, effectively DoS'ing the withdrawal.
Recommendations	We recommend simply removing the totalUser logic.
Comments	

Issue	The depositFromToken function is flawed
Severity	**High**
Description	The depositFromToken function is flawed
Recommendations	Call _deposit with the received ETH amount as parameter
Comments	

Issue	Inefficient deposit/withdrawal architecture
Severity	**High**
Description	During each deposit and withdrawal, users may undergo token swaps between the LP token/BNB and vice versa. These swaps are exclusively executed through the pancakeswap pairs, inevitably incurring slippage costs borne by the users. Particularly for

	substantial deposit positions, users are at risk of experiencing financial losses. Furthermore, this architectural design exposes the system to potential devastating sandwich attacks.
Recommendations	We strongly recommend rethinking this architecture.
Comments	

Issue	Users might lose tokens during withdraw
Severity	**High**
Description	The issue arises when a user withdraws tokens, as the user_deposits[msg.sender] mapping is set to zero. However, this also occurs when the user does not withdraw the entire amount, resulting in a loss of funds for the user during subsequent withdrawals. Proof of Concept (PoC): 1. Alice stakes 100 tokens. 2. Alice withdraws 10 tokens, which sets Alice's user_deposits to zero: user_deposits[msg.sender] = 0; 3. Alice then withdraws the remaining 90 tokens. Since user_deposits is now zero, Alice will incur a 30% fee on the 90 tokens, even though the fee should only be applied to the accumulated reward.
Recommendations	Deduct user deposits only by the amount parameter.
Comments	

Issue	Users can circumvent the performance fee
Severity	**High**

Description	The _withdraw function incorporates logic intended to levy users a performance fee based on their profits. This issue will be addressed independently from the previous issue mentioned.
	Users have the ability to bypass this check due to the manner in which the increase of user_deposits is implemented as follows:
	
	user_deposits[msg.sender] += amount;
	
	However, the user can determine whether amount represents BNB or LP tokens. During the withdrawal process (once the flaw is rectified), the user can specify whether it is via BNB or LP tokens by assigning the following:
	withdrawAmount = bnbAmount;
	or
	withdrawAmount = amount;
	As a result, a highly sophisticated user could, depending on the value of LP tokens compared to BNB, exclusively deposit LP tokens and subsequently withdraw BNB. This would lead to the calculation of profits based on the nominal amounts of deposited LP tokens and withdrawn BNB, effectively enabling the user to circumvent the performance fee.
	
	It is important to note that this issue will also manifest in regular business logic and can potentially cause users to experience financial losses if the process is reversed.
	
	In summary, it is advised to avoid comparing different currencies for this particular check.

Recommendations	Addressing this issue requires a significant amount of effort and ingenuity. One potential solution is to restrict users to only withdraw funds in the same currency as their initial deposit. Alternatively, an alternative approach involves implementing a mechanism that stores

	the deposits and their corresponding currencies. Subsequently, calculations can be performed to determine the USD-value of the deposits, and a similar process can be applied to withdrawals.
Comments	

Issue	restakeThreshold logic might get abused
Severity	High
Description	The restakeThreshold variable is set by the owner and determines when a compound should happen and when not. A malicious user could inspect the current pending rewards and execute a deposit whenever the amount is below the threshold. This will result in rewards being harvested but not compounded or used in any way, essentially stucking in the contract forever.
Recommendations	Remove the restakeThreshold logic or only allow it being set up to 1e16 tokens for example in order to make this attack economically uninteresting. We recommend following the latter recommendation in order to prevent compounding if no tokens have been staked before.
Comments	

Issue	Frontrun of withdrawToToken
Severity	High
Description	The aforementioned function can be frontran by a malicious user in an effort to increase profit from the users swap, resulting in a loss for the user.
Recommendations	This issue has been noted in the DEX contract as well, however, due its high impact it will be emphasized also here.
Comments	

Issue	Contract does not work with transfer-tax tokens
Severity	High
Description	The contract does not account for transfer-tax tokens. Several functions will just revert.
Recommendations	Do not allow routing through/from/to these tokens or implement accounting logic for it. We are fine with having this issue acknowledged.
Comments	

Issue	Lack of emergencyWithdraw
Severity	Medium
Description	While we consider the pancakeswap staking contract as highly trustable, there still might be some edge-cases that can result in the necessity of withdrawing funds via emergencyWithdraw
Recommendations	Implement emergency withdrawal logic that can be executed by the admin.
Comments	

Issue	Contract does not use safeApprove
Severity	Low
Description	The contract is using the regular approve function, this does not work for tokens that do not conform to ERC20 standards ie USDT on mainnet. In an effort to achieve the highest possible compliance, the approval process should be as follows:

	token.safeApprove(xx, value) execute approval consuming function token.safeApprove(xx, 0)
Recommendations	Follow the aforementioned example to achieve highest compliance.
Comments	

Issue	No option to withdraw to LP
Severity	Medium
Description	Currently, the contract has no option for a user to simply withdraw the LP pair, it might be valuable for users to do so in order to circumvent potential slippage costs. We assume that this was already intended to implement due to the _withdraw function having the correct parameter - however it stays unused
Recommendations	Add a function which allows the user to withdraw solely the LP token.
Comments	

Issue	stakeStableCoinRewards will accumulate dust
Severity	Medium
Description	The function mentioned above follows the following logic: 1) Calculate the share for LP creation. 2) Execute the zap to the desired LP token. However, in the case of pairs involving BNB, the following steps are performed: 1) Swap half of the BNB for the token. 2) Add the remaining half of BNB along with the amount obtained from the previous swap to the liquidity. Regrettably, the initial swap always incurs slippage, resulting in a lower amount of BNB being utilized for the pair. Consequently, a remaining amount of BNB is sent back to the Cairo contract and becomes permanently locked within it
Recommendations	Add a function that allows the admin to withdraw the accumulated BNB
Comments	

Issue	withdraw might not work for smart contracts
Severity	Informational
Description	The withdraw function transfers BNB back to the caller, however, if that's a smart contract without a fallback function, it will simply not work.
Recommendations	At this point there is not much one can do besides advising these cases to use withdrawToToken.
Comments	

Issue	Minor issues
Severity	Informational
Description	The codebase exhibits minor issues, as outlined below: 1. The import statement for SafeMath can be removed since solidity 0.8.4 is being used. 2. The import statement for SafeToken is unused. 3. The StableCoinStrategy is not initialized. 4. The variable totalStandardBnbReinvested is unused. 5. The check for isNotPaused() is redundant as the pausable library already ensures this. The same issue applies to unpause. 6. The variables earnedHoney, earnedBnb, and stakedHoney are assigned the value 0 but are not used. 7. The comment mentions the use of AccessControl from the OpenZeppelin implementation to handle the update of the beeEfficiency level, but there is no beeEfficiency level in the code. 8. The parameter description for amount inaccurately states that only BNB can be deposited, while users can also deposit LP tokens.
Recommendations	Fix the issues above.

6. DEX

The DEX router plays a significant role in the Cairo contract, facilitating token swaps before each deposit and after each withdrawal. The Cairo contract enables users to deposit both BNB and arbitrary tokens, which are then swapped to LP tokens by the DEX contract. During a withdrawal, users have the option to receive either BNB or an arbitrary token, and the DEX performs the necessary token swaps within the LP pair to provide the desired output.

The DEX contract provides the following functionalities:

1) Zapping BNB to LP tokens:
   a) LP tokens including WBNB
   b) LP tokens without WBNB

2) Unzapping LP tokens to BNB:
   a) LP tokens including WBNB
   b) LP tokens without WBNB

3) Swapping BNB to tokens

4) Swapping tokens to BNB

Additionally, users can fetch their pending rewards and access information regarding the number of LP tokens they will receive for their pending rewards.

This contract is a modified fork of Grizzly Finance.

Issue	Architectural Risk: Arbitrary paths
Severity	Architecture
Description	As below mentioned, the UPDATER can set all token paths for swap and liquidity operation processes This setup might result in large issues even for the smallest misconfiguration.
Recommendations	While we realize the necessity of this setup, we highly recommend setting up all paths with the highest level of caution.
Comments	

Issue	UPDATER can set arbitrary paths
Severity	Governance
Description	In order to swap BNB -> token or vice-versa, a specific path must have been set. The updater role can set any arbitrary path, including a path to a dummy token only he owns. Which means that he can essentially steal all funds during a swap.
Recommendations	Use a trusted multi-sig wallet as updater address, implement a time lock feature for this function.
Comments	

Issue	PAUSER can block withdrawals and deposits
Severity	Governance
Description	The PAUSER role can pause all swap and liquidity operations, essentially preventing users from withdrawing their funds
Recommendations	Use a trusted multi-sig wallet as a pauser address.
Comments	

Issue	Contract does not work with transfer-tax tokens
Severity	High
Description	The contract does not account for transfer-tax tokens. Several functions will just revert.
Recommendations	Do not allow routing through/from/to these tokens or implement accounting logic for it. We are fine with having this issue acknowledged.
Comments	

Issue	An attacker can frontrun swaps
Severity	High
Description	All swapping functions are using 1 as minAmountOut. This can be brutally abused by other users/MEV bots up to a point where the depositor/swapper loses all of their funds.
Recommendations	Allow the user to input a minAmountOut as a parameter. It is important to allow the user to do this because an internal calculation within the same transaction would be redundant. The simplest solution is to execute the slippage check before the swap as GrizzlyFi is doing (this must be done in the Cairo contract).
Comments	

Issue	Sub-optimal contract logic
Severity	Medium
Description	The contract is using the PancakeSwap liquidity pools in order to swap different tokens to bnb or vice-versa. While we accept that this logic works for swapping and compounding rewards, we discourage the use for zapping liquidity in and out. Depending on the depth of the liquidity pool, users will always lose tokens when zapping in and out. If for example a user wants to deposit a huge amount of tokenX/tokenY and zaps in with BNB, the user will encounter huge slippage, depending on the deposited amount and the depth of the liquidity pool.
Recommendations	Use 1inch for swaps or remove the zap logic completely
Comments	

Issue	totalPendingRewards is flawed for multipath swaps
Severity	Medium
Description	The totalPendingRewards function returns the wrong amountOut for multipath swaps: uint256 tokenAValue = SwapRouter.getAmountsOut(pendingRewardToken / 2, pairsTokenA)[2]; This code section assumes that the length of the swap path is always three which means that the second index will always become the result output, however, for multipath swaps this won't work.
Recommendations	Use pairsTokenA/B.length -1 instead of 2.
Comments	

Issue	getTokenEthPrice should never be used.
Severity	Medium
Description	The aforementioned function simply determines how many tokens a user would receive for 1 eth depending on one pool - however, this can be manipulated easily in order to manipulate the return value of this function. Depending on how this function is used within external contracts this issue can easily escalate to high risk. Moreover, the admin can change the path anytime which increases the risk even more if external contracts rely on it.
Recommendations	Use a chainlink oracle whenever figuring out prices. First fetch the price in USD and then use the ETH price feed in order to calculate the price in ETH.
Comments	

Issue	checkSlippage function has a minor flaw
Severity	Low
Description	The checkSlippage function determines if an output amount is within an acceptable slippage range. However, this check is slightly flawed. require(((MAX_PERCENTAGE - slippage) * amountOut[i]) / MAX_PERCENTAGE < currentAmoutOut); The function will also revert if currentAmountOut is equal to the calculation, however, in fact the slippage is still valid if it is equal.
Recommendations	Change the equation to also allow for the equal case.
Comments	

Issue	Contract does not use saveApprove
Severity	Low
Description	The contract is using the regular approve function, this does not work for tokens that do not conform to ERC20 standards ie USDT on mainnet. In an effort to achieve the highest possible compliance, the approval process should be as follows: token.safeApprove(SwapRouter, value) SwapRouter.call... token.safeApprove(SwapRouter, 0)
Recommendations	Follow the aforementioned example to achieve highest compliance.
Comments	

Issue	Redundant refund architecture
Severity	Informational
Description	During the addLiquidity function, the router will always quote the correct amounts, this means that one of both tokens will be fully consumed whereas the co token amount will be decreased in order to match the ratio in the pair. This means that there is always only one token leftover to refund. However, within the convertEthToPairLp function, both tokens will be refunded, ignoring the fact that only one of both tokens has a valid leftover amount.
Recommendations	Implement logic to only refund the token with the actual leftover amount.
Comments	

Issue	Deadline is unnecessarily increased
Severity	Informational
Description	For in-transaction swaps it is unnecessary to set a deadline larger than block.timestamp since the swap is in fact within the same block executed.
Recommendations	Simply use block.timestamp as the deadline.
Comments	

Issue	Uninitialized dependencies
Severity	Informational
Description	The ReentrancyGuardUpgradeable, AccessControlUpgradeable and PausableUpgradeable dependencies are uninitialized.
Recommendations	Initialize the aforementioned dependencies.
Comments	

Issue	__gap can be removed
Severity	Informational
Description	A gap variable is only necessary for contracts that are meant to be inherited because their storage variables will be initialized first and an addition would therefore result in storage collision. However, for the main contract it is not necessary since future variables can just be added below the latest variable.
Recommendations	Remove the gap variable.
Comments	

7. Referral Contract

The ReferralContract serves as an external contract that is invoked by the Cairo contract to handle referral-related logic. Its functionality is outlined as follows:

1) Referral Assignment: Each depositor can be associated with a single referrer. When a withdrawal occurs, a performance fee is applied, which is subsequently distributed to the referrer.

2) Upstream Referral Rewards: Additionally, the referrer's referrer is eligible to receive a reward, creating a potential upstream distribution of rewards up to four referrers in total. Each level of distribution is uniquely defined and configured within the initialization function, with the combined distributions summing up to 100 to align with the divisor.

3) Handling Unreferenced Addresses: If an address does not have any upstream referrer, the entire remaining amount after fee deduction will be distributed to the original withdrawal address.

The ReferralContract utilizes OpenZeppelin's AccessControl library to facilitate privileged interactions. The administrator has the ability to add any address as a REWARDER, which ideally should be limited to the Cairo contract for appropriate access.

To avoid potential collisions and ensure proper segregation, it is recommended to deploy a new instance of the ReferralContract for each strategy contract, mitigating any possible interference or conflicts between different strategies.

Issue	The referral logic is flawed
Severity	**High**
Description	The referral logic in the system has a flaw that renders it ineffective in achieving the desired outcome. This issue can be demonstrated through the following Proof of Concept (PoC): 1. Alice performs a token deposit in the vault and specifies Bob as the referrer. 2. This action updates the mapping as follows: referralGiverAddresses[ALICE] = BOB.

3. The next step involves incrementing Bob's count of active friends:
activeFriends[_referralGiver] += 1.

4. Subsequently, the giver address is fetched from the receiver address:
address _constRefereralGiver = referralGiverAddresses[ALICE].
This results in obtaining Bob's address.

5. Finally, the referralGivers mapping is assigned as follows:
referralGivers[CAIRO][BOB] = BOB.

As can be observed, Bob is assigned to himself in this step, while the expectation is for Bob to be assigned to Alice. Despite this issue, let us continue the PoC with Alice initiating a withdrawal.

---

6. Alice calls the withdraw function, which retrieves the following mapping:
address _referralGiver = referralGivers[CAIRO][ALICE].

Unfortunately, due to the flawed assignment in step 5, where Bob is assigned to Bob, Alice is not referring to any person. Consequently, all funds will be sent to Alice instead of Bob.

.....

Now, let us replay the PoC with Bob calling the withdraw function:

6. We retrieve the following mapping:
address _referralGiver = referralGivers[CAIRO][BOB].

The return value of this mapping will be Bob's address, indicating that Bob will receive all the rewards.

This PoC serves to illustrate the issue with the referral logic, where incorrect assignments and the distribution of rewards occur.

Recommendations	The correct implementation would be to set Bob as Alice's referral giver: referralGivers[msg.sender][_referralRecipient] = _constRefereralGiver; Moreover, we do not understand why the last line within the referralDeposit function is executed each time as it should only be set when the referralRecipient not has an assigned referrer yet
Comments	

Issue	BNB transfer will always revert for smart contracts
Severity	**High**
Description	The BNB transfer is currently done via a simple transfer: payable(to).transfer(amount); However, this will not work for smart contracts as to address, resulting in stuck funds within the referral contract.
Recommendations	An easy fix would be to use .call instead of .transfer.
Comments	

Issue	Possible referral collisions
Severity	**Medium**
Description	The referralDeposit function implies that the referral contract is utilized by multiple strategies, as evidenced by the line of code: referralGivers[msg.sender][_constRefereralGiver] = _constRefereralGiver;

	Based on the usage of the msg.sender variable, we make this assumption. However, this approach may lead to collisions since all upstream referrers rely on the following mapping: _referralGiver = referralGiverAddresses[_referralGiver]; It is important to note that the value of the _referralGiver variable is only assigned during the initial deposit made by the referralRecipient. Additionally, employing multiple rewarder contracts increases the potential for cross-contract reentrancy attacks.
Recommendations	We highly recommend deploying a separate referral contract for each strategy in order to avoid any unexpected side-effects.
Comments	

Issue	Unchecked initialization
Severity	Medium
Description	The contract initializes the following variables: _admin, _devTeam, and _ReferralRewards. However, there are no checks in place to ensure the correctness of these variables. Specifically: ● _admin should not be set to address(0). ● _devTeam should not be set to address(0) or a contract without a fallback function. ● _ReferralRewards should be an array consisting of four uint values, with an aggregated value of 100.
Recommendations	Implement a check for these variables
Comments	

Issue	Missing nonReentrant modifier
Severity	Medium
Description	During the referralWithdraw function, ether is sent to specific addresses, this can lead to undesired reentrancy calls, specifically if multiple rewarders are allocated to this contract which should however never be the case
Recommendations	Implement nonReentrant to the referralWithdraw function.
Comments	

Issue	Anyone can initialize the implementation
Severity	Low
Description	This contract can be initialized by anything which might be abused by tricking users to send ether to it in order to obtain it.
Recommendations	Disable the functionality for direct initialization during the contract deployment.
Comments	

Issue	Unclean codebase
Severity	Informational
Description	The codebase contains several unused sections that do not serve any purpose and can be safely removed: 1. import "@openzeppelin/contracts/utils/math/SafeMath.sol"; 2. import "@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol"; 3. ReentrancyGuardUpgradeable 4. bytes32 public constant PAUSER_ROLE =

	keccak256("PAUSER_ROLE"); 5. bytes32 public constant UPDATER_ROLE = keccak256("UPDATER_ROLE"); 6. mapping(address => uint256) private totalReferralDeposits; 7. event RewardsClaimed(address indexed _user, uint256 rewards); 8. IERC20Upgradeable private CairoToken;
Recommendations	Cleanup codebase from unused sections mentioned above.
Comments	

9. SafeToken

SafeToken contract is responsible for safely transferring ETH and ERC20 tokens, the base logic is forked from the SafeERC20 library.

No issues.