PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Preliminary Report

For KogeFarm's V2 Vaults

10 September 2021

paladinsec.co

info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

1 Overview

This report has been prepared for the Koge Farm's V2 Vaults on the Polygon network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	KogeFarm's V2 Vaults
URL	https://kogefarm.io
Platform	Polygon
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`vaultBase`	`0xe47b2E9c02EF23B91cdfc1ADbfA9bBb4372d421B`	MATCH
`IStrategy`	`Dependency`	MATCH
`strategyBase`	`0x998c465ef2285e7c3534ad253188659767f341a8`	MATCH
`BaseStrategyMasterChef`	`0x998c465ef2285e7c3534ad253188659767f341a8`	MATCH
`StrategyTwoAssets`	`0x998c465ef2285e7c3534ad253188659767f341a8`	MATCH
`StrategyFarmTwoAssets`	`0x998c465ef2285e7c3534ad253188659767f341a8`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0	-	-	-
Medium	4	4	-	-
Low	4	4	-	-
Informational	24	23	-	1
Total	**32**	**31**	**-**	**1**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 jarBase and vaultBase

ID	Severity	Summary	Status
01	MEDIUM	`depositAll` and `withdrawAll` do not work due to redundant reentrancy guards requiring users to always use the `deposit` and `withdraw` methods	RESOLVED
02	MEDIUM	No check is done to ensure the deposit fee of the underlying Masterchef is within reasonable bounds	RESOLVED
03	LOW	While smart contracts are not allowed to deposit into a vault, they are still allowed to withdraw from it	RESOLVED
04	INFO	The `onlyEOA` check can be expanded to be more redundant in case the EVM evolves	RESOLVED
05	INFO	Inconsistent data size when calling `poolInfo`	RESOLVED
06	INFO	Inconsistencies in token casting	RESOLVED
07	INFO	Behavior specific to individual strategies is hardcoded within the JarBase which requires adopting the JarBase to individual strategies	RESOLVED
08	INFO	Jar deposits are inefficient for tokens with a transfer tax	RESOLVED
09	INFO	Jar deposits cannot be paused	RESOLVED
10	INFO	Gas optimization: Redundant math during withdraw call	RESOLVED
11	INFO	`token` and `strategy` can be made `immutable`	RESOLVED
12	INFO	`getRatio` and `getLastTimeStaked` can be made `external`	RESOLVED
13	INFO	Lack of space in the share token name	RESOLVED
14	INFO	Lack of events for the `deposit` and `withdraw` functions	RESOLVED

1.3.2 IStrategy

ID	Severity	Summary	Status
15	INFO	Gas optimization: Usage of smaller sized integers like `uint16` is not gas efficient	RESOLVED

1.3.3 BaseStrategy

ID	Severity	Summary	Status
16	LOW	Gov privilege: Setting `strategist` to the zero address will break most functionality	RESOLVED
17	INFO	Gas optimization: Unnecessary and inconsistent timestamp increment during Uniswap transactions	RESOLVED
18	INFO	Unused internal function `withdrawAll`	RESOLVED
19	INFO	`want` and `harvestedToken` can be made `immutable`	RESOLVED
20	INFO	Lack of events for the `setStrategist` function	RESOLVED

1.3.4 BaseStrategyMasterchef

ID	Severity	Summary	Status
21	MEDIUM	Tokens with a transfer tax cannot be withdrawn	RESOLVED
22	LOW	Lack of reentrancy guards on important functions	RESOLVED
23	INFO	`rewards` and `poolId` can be made `immutable`	RESOLVED
24	INFO	`salvage` and `emergencyWithdraw` can be made `external`	RESOLVED
25	INFO	Lack of events for the `set_fee`, `set_multiHarvest`, `set_harvestCutoff` and `harvest` functions	RESOLVED

1.3.5 StrategyTwoAssets and StrategyFarmTwoAssets

ID	Severity	Summary	Status
26	MEDIUM	Qi DAO Masterchef Staking Pool does not have an earned function	RESOLVED
27	LOW	Fee mechanism does not support transfer tax rewards in case the `feeTokenAddr` is set to a different token	RESOLVED
28	INFO	Uniswap routing paths might be inefficient	RESOLVED
29	INFO	Error message still references "Titan"	RESOLVED
30	INFO	`wantToken` should never be set to the `feeTokenAddr`	RESOLVED
31	INFO	Token symbol exceeds 11 characters which makes adding it to MetaMask more cumbersome	ACKNOWLEDGED

2 Findings

2.1 jarBase and vaultBase

Within the Koge vault system, a Jar represents a vault. It is backed by a single token that it compounds over time and emits a share token which represents ownership of the Jar. This is backed by a strategy contract which generates revenue. It should be noted that the strategy contract cannot be upgraded so there is significantly less governance risk with Koge compared to alternative vaults.

Finally, it should be noted that any code that was commented out was not included within the scope of this audit and this section of the audit mainly focuses on how the JarBase interacts with the StrategyTwoAssets which is included in this audit.

2.1.1 Issues & Recommendations

Issue #01	**depositAll and withdrawAll do not work due to redundant reentrancy guards requiring users to always use the deposit and withdraw methods**
Severity	MEDIUM SEVERITY

Location	Lines 58-62
	`function depositAll() external nonReentrant {`
	`  deposit(token.balanceOf(msg.sender));`
	`}`
	`function deposit(uint256 _amount) public nonReentrant {`
	
	Lines 104-109
	`function withdrawAll() external nonReentrant {`
	`  withdraw(balanceOf(msg.sender));`
	`}`
	
	`// Withdraw some balances`
	`function withdraw(uint256 _shares) public nonReentrant {`

Description	The JarBase contract contains two useful utility functions `depositAll` and `withdrawAll`. The `depositAll` function will deposit the whole user balance to the vault while `withdrawAll` will withdraw the whole position the user has within the vault. However, due to redundant `nonReentrant` modifiers, the `depositAll` (and respectively `withdrawAll`) call will already have locked the contract by the time `deposit` (respectively `withdraw`) is called. Since the `deposit` and `withdraw` functions contain a second pair of `nonReentrant` calls, these will revert as the reentrancy lock is active at that point.
Recommendation	Consider removing the redundant reentrancy guards on `depositAll` and `withdrawAll` since they are already present on the public `deposit` and `withdraw` functions. These latter ones should of course be retained.
Resolution	RESOLVED The `nonReentrant` checks have been removed from the `depositAll` and `withdrawAll` functions.

Issue #02	No check is done to ensure the deposit fee of the underlying Masterchef is within reasonable bounds
Severity	MEDIUM SEVERITY
Description	Although it does not happen often, sometimes malicious protocols set the deposit fees of their Masterchef to 100% or higher. There is no check to ensure that deposits revert when this happens, which means investors could end up receiving no shares in return when they deposit funds into the protocol.
Recommendation(s)	Consider validating that the actual shares received are within a percentage bound of the shares that were requested. In case the Masterchef remains hardcoded within the protocol, a simple check on the `depositFee` suffices. `require(depositFee <= MAX_DEPOSIT_FEE_MC, "!MC DEPOSIT FEE");`
Resolution	RESOLVED A generic check to ensure that the user receives at least 90% of the deposited value is now made. This check works on all strategies.

Issue #03	**While smart contracts are not allowed to deposit into a vault, they are still allowed to withdraw from it**
Severity	LOW SEVERITY
Description	The client has implemented an extra safety guard on the deposit function to prevent smart contracts from making deposits. This is presumably done to prevent exploits since often an exploit requires to be executed within a single transaction. However, this safety measure has not been taken in the `withdraw` function. Although it might seem as if no smart contract can withdraw since they did not deposit, this is not true because shares can be transferred to a smart contract. Because of this reason, the severity of this issue has been increased from informational to low.
Recommendation	We are generally no advocates of adding EOA-only restrictions since it hinders composability, however, in this case this is seen as inconsistent and we recommend adding an `onlyEOA` check to withdrawals as well.
Resolution	RESOLVED The client has pointed out that they always want to allow people to withdraw, and since it is good practice to assume that `tx.origin` might break on future hardforks, this was the best solution.

Issue #04	**The onlyEOA check can be expanded to be more redundant in case the EVM evolves**
Severity	INFORMATIONAL
Location	Line 63 `require(msg.sender == tx.origin, "no contracts");`
Description	Although at Paladin we are in general not fans of the usage of `tx.origin` when it is used, we recommend making the check redundant with the OpenZeppelin `isContract` address check. This is because if the `tx.origin` check ever loses its significance through protocol upgrades (which all though extremely unlikely, should be assumed), the OpenZeppelin check will still serve as a second safety guard.
Recommendation	Consider adding an onlyEOA modifier which includes the OpenZeppelin Address contract check: `modifier onlyEOA() { require(msg.sender == tx.origin && !address(msg.sender).isContract(), “no contracts”); }`
Resolution	RESOLVED The check has been extended with the recommendation.

Issue #05	Inconsistent data size when calling `poolInfo`
Severity	INFORMATIONAL
Location	Line 86 `(, , , , uint256 depositFee) = IMasterChef(masterChefAddr).poolInfo(underlyingPoolId);`
Description	Most `poolInfo` structs on a Masterchef have a `depositFee` as an `uint16` type; however, within the call at line 86, it is stored inside a `uint256`. Although we believe Solidity will simply cast this without any problem, this might have been done by accident without understanding the underlying interface and could potentially put off third-party reviewers when they see this code.
Recommendation	Consider following the interface when interpreting the `poolInfo`. In case the implicit cast was intentional and has been tested, nothing has to be done.
Resolution	RESOLVED The Masterchef behavior has been removed completely to resolve the tight coupling issue.

Issue #06	Inconsistencies in token casting
Severity	INFORMATIONAL

Location	Lines 39-40
	`_setupDecimals(ERC20(_strategy.want()).decimals());`
	`token = IERC20(_strategy.want());`

Description	Although completely harmless, throughout the code, sometimes token addresses are cast to ERC20 and other times IERC20 is used. This is a simple inconsistency which should not be necessary.
Recommendation	Consider always using IERC20 to be consistent throughout the codebase.
Resolution	RESOLVED The client used ERC20 as a way to get the decimals function which is only defined in IERC20MetaData, and casting the first variable to IERC20MetaData might even be less readable hence this issue is marked as resolved.

Issue #07	**Behavior specific to individual strategies is hardcoded within the JarBase which requires adopting the JarBase to individual strategies**
Severity	INFORMATIONAL
Description	The JarBase explicitly reads the Masterchef functionality of the underlying strategy, which requires each strategy with a different Masterchef signature to have its own unique JarBase code. This is undesirable as the whole point of JarBase is to extract the code which should not have to be changed when strategies evolve.
Recommendation	The only place where the Masterchef is used within the JarBase is to calculate the deposit fee size. This deposit fee can also be calculated through a before-after check of the strategy balance. If such a change would be implemented, the `rewards()` variable no longer have to be called and `underlyingPoolId` can be removed from the IStrategy interface. It should be noted that in this case, the `deposit` function in IStrategy should be changed from `deposit()` to `deposit(uint256 amount)` because if it does a full deposit, it will also stake any coins that are sent manually to the strategy which will have a negative effect on the before-after check and could lead to DoS. Finally, the `withdrawPenalty` function is not present on any Masterchefs known to us and it is not used within the contract, this function can thus be removed from the IMasterchef interface in case the interface is kept.
Resolution	RESOLVED The Masterchef behavior has been removed.

Issue #08	Jar deposits are inefficient for tokens with a transfer tax
Severity	INFORMATIONAL
Description	Currently when a deposit is made, the tokens are first sent to the Jar, and then to the strategy. This requires an extra transfer when this is not strictly necessary and thus the transfer tax is incurred twice for deposit fee tokens.

Recommendation	Consider transferring the tokens directly to the strategy after ensuring that this would have no side-effects.
	`uint256 _before = token.balanceOf(strategy);`
	`token.safeTransferFrom(msg.sender, strategy, _amount);`
	`uint256 _after = token.balanceOf(strategy);`

Resolution	RESOLVED The tokens are now directly sent to the strategy.

Issue #09	Jar deposits cannot be paused
Severity	INFORMATIONAL
Description	Currently there is no way for the Jar governance to pause deposits – this might be desired if the underlying protocol has issues.
Recommendation	Consider implementing `Pausable` and adding `whenNotPaused` to the `deposit` function.
Resolution	RESOLVED The contract is now pausable and only deposits are prevented while it is paused.

Issue #10	Gas optimization: Redundant math during withdraw call
Severity	INFORMATIONAL

Location	Lines 116-122
	`uint256 _withdraw = r.sub(b);`
	`IStrategy(strategy).withdraw(_withdraw);`
	`uint256 _after = token.balanceOf(address(this));`
	`uint256 _diff = _after.sub(b);`
	`if (_diff < _withdraw) {`
	`  r = b.add(_diff);`
	`}`

Description	Within the `withdraw` function, `r` is set to `b + _diff`. However, this code simplifies to `_after`.

Recommendation	Consider setting `r` to `_after` directly to save some gas for the redundant addition.
	`if (_diff < _withdraw) {`
	`  r = _after;`
	`}`

Resolution	RESOLVED The recommended simplification has been made.

Issue #11	`token` and `strategy` can be made `immutable`
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the `immutable` keyword. This is considered best practice since it makes the code more accessible for third-party reviewers. Furthermore it can result in lower gas usage since these variables are hardcoded in the bytecode.
Recommendation	Consider making the above variables explicitly `immutable`.
Resolution	RESOLVED

Issue #12	`getRatio` and `getLastTimeStaked` can be made `external`
Severity	INFORMATIONAL
Description	The contract contains functions that can be changed from `public` to `external`. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider making the above functions external.
Resolution	RESOLVED

Issue #13	Lack of space in the share token name
Severity	INFORMATIONAL
Location	Line 159 `string(abi.encodePacked(_strategy.pairName(), "vault")),`
Description	When the `pairName` is for example set to `BUSD_USDC` the share token will be called `BUSD_USDCvault`.
Recommendation	Consider whether this is desired, otherwise consider adding a space between the two components of the share name so the example would become `BUSD_USDC vault`. `string(abi.encodePacked(_strategy.pairName(), " ", "vault"));`
Resolution	RESOLVED

Issue #14	Lack of events for the `deposit` and `withdraw` functions
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	RESOLVED

2.2 IStrategy

The IStrategy interface defines the functions every strategy needs to implement.

2.2.1 Issues & Recommendations

Issue #15	Gas optimization: Usage of smaller sized integers like uint16 is not gas efficient
Severity	INFORMATIONAL
Location	Line 14 `function underlyingPoolId() external view returns(uint16);`
Description	Within the current Solidity compiler, smaller types like `uint16` actually consume slightly more gas then the standard `uint256` type. This is because under the hood, the EVM uses 256 bits for storage slots. Smaller types actually need some extra conversion logic contrary to the wider `uint256`.
Recommendation	Consider always using `uint256` in favor of smaller types, and limits can be explicitly set through `require` statements.
Resolution	RESOLVED The function has been removed from the interface completely.

2.3 BaseStrategy

The BaseStrategy is a dependency included in the actual strategy that defines some basic functionality for the eventual strategy like functions to swap tokens.

It should be noted that the Jar can deposit into the strategy even when the strategy is not yet linked to the Jar, however withdrawals are not possible in this case. Users should thus take a quick look that the strategy is actually linked to the Jar. We will do this as well for any strategies that are verified in this report.

2.3.1 Issues & Recommendations

Issue #16	**Gov privilege: Setting strategist to the zero address will break most functionality**
Severity	LOW SEVERITY
Description	During harvests, a variety of tokens are sent to the `strategist` account. However, most tokens revert if the destination is the zero address and thus if this strategist account is ever set to the zero address, harvesting functionality will break.
Recommendation	To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like: `require(_strategist != address(0), "!nonzero");` to the configuration function.
Resolution	RESOLVED The check has been added to both locations.

Issue #17	Gas optimization: Unnecessary and inconsistent timestamp increment during Uniswap transactions
Severity	INFORMATIONAL

Location	Lines 100-107
	`IUniswapRouterV2(dexRouter).swapExactTokensForTokens(`
	`  _amount,`
	`  0,`
	`  path,`
	`  address(this),`
	`  now.add(600)`
	`);`
	
	Line 150 (inconsistency)
	`now + 600`

Description	When Uniswap interactions are done, 600 seconds are added to the deadline parameter. This is not necessary since now is a synonym for `block.timestamp` and this timestamp does not change throughout the transaction. A deadline of simply now is thus sufficient. Furthermore, on line 150, the addition is done without using SafeMath. Although we see absolutely no reason why this would overflow or even cause any issues if it overflows, consistent code signals to third-party reviewers that the codebase has been carefully considered and is thus always recommended.
Recommendation	Consider replacing the deadline parameters to simply now.
Resolution	RESOLVED All deadlines are adjusted to use now.

Issue #18	**Unused internal function `withdrawAll`**
Severity	INFORMATIONAL
Location	Line 80 `function _withdrawAll() internal {`
Description	The code contains an unused internal function `_withdrawAll`. Including unused functionality makes third-party reviews unnecessarily more complex.
Recommendation	Consider removing the unused functionality.
Resolution	RESOLVED The function is now removed.

Issue #19	**`want` and `harvestedToken` can be made `immutable`**
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the `immutable` keyword. This is considered best practice since it makes the code more accessible for third-party reviewers. Furthermore, it can result in lower gas usage since these variables are hardcoded in the bytecode.
Recommendation	Consider making the above variables explicitly `immutable`.
Resolution	RESOLVED

Issue #20	Lack of events for the `setStrategist` function
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above function.
Resolution	RESOLVED

2.4 BaseStrategyMasterchef

The BaseStrategyMasterchef contract contains all relevant code for a strategy that deposits tokens into a MasterChef. The compounding logic still needs to be included in a contract that inherits this base to account for either LP compounding or simple single-asset compounding. As we were only provided with code that does LP compounding (StrategyTwoAssets) we have audited the code mainly with this functionality in mind.

The contract contains a salvage governance function which allows taking out any token but the reward and staking token. This function should thus be harmless.

2.4.1 Issues & Recommendations

Issue #21	Tokens with a transfer tax cannot be withdrawn
Severity	MEDIUM SEVERITY

Location	Lines 238-239
	`IMasterChef(rewards).withdraw(poolId, _amount);`
	`return _amount;`

Description	Throughout the codebase, the system has tried to account for transfer tax tokens to some extent. However, within the `_withdrawSome` function, the amount requested from the masterchef is blindly forwarded. In case there is a transfer-tax on said token, the strategy will not receive the whole amount, the same goes for potential withdrawal fees.

Recommendation	Consider using a before-after pattern within the `_withdrawSome` function.
	`int256 _before = IERC20(want).balanceOf(address(this));`
	`IMasterChef(rewards).withdraw(poolId, _amount);`
	`return _before.sub(IERC20(want).balanceOf(address(this)));`
	Note that any function that calls this `withdrawSome` function should be guarded against reentrancy as should always be done with a before-after pattern.

Resolution	RESOLVED A before-after withdrawal is now done.

Issue #22	Lack of reentrancy guards on important functions
Severity	LOW SEVERITY
Description	All though we could not find any exploits ourselves, we recommend adding reentrancy guards to `deposit()`, `salvage()` and `emergencyWithdraw()`.
Recommendation	Consider validating the Masterchef interfaces with all functions from the Qi DAO Masterchef and renaming the function to `pending`.
Resolution	RESOLVED `jarDeposit`, `salvage` and `emergencyWithdraw` now all have `reentrancyGuards`.

Issue #23	`rewards` and `poolId` can be made `immutable`
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the `immutable` keyword. This is considered best practice since it makes the code more accessible for third-party reviewers. Furthermore, it can result in lower gas usage since these variables are hardcoded in the bytecode.
Recommendation	Consider making the above variables explicitly `immutable`.
Resolution	RESOLVED

Issue #24	`salvage` and `emergencyWithdraw` can be made external
Severity	INFORMATIONAL
Description	The contract contains functions that can be changed from `public` to `external`. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider making the above functions external.
Resolution	RESOLVED

Issue #25	Lack of events for the `set_fee`, `set_multiHarvest`, `set_harvestCutoff` and `harvest` functions
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	RESOLVED

2.5 StrategyTwoAssets and StrategyFarmTwoAssets

The StrategyTwoAssets is a compounding strategy contract to harvest and compound rewards for the miMATIC/USDC pair within the Qi DAO Masterchef Staking Pool. The parent StrategyTwoAssets contract is deployable.

A fee which can be configured up to at most 10% is taken from every harvest.

2.5.1 Issues & Recommendations

Issue #26	Qi DAO Masterchef Staking Pool does not have an `earned` function
Severity	MEDIUM SEVERITY
Location	Line 218 `return IMasterChef(rewards).earned(poolId, address(this));`
Description	Within the `getHarvestable view` function (which is only used for UI purposes), the earned method is called on the Masterchef. However, within the Qi DAO Masterchef, this function is called `pending`.
Recommendation	Consider validating the Masterchef interfaces with all functions from the Qi DAO Masterchef and renaming the function to pending. Note that the IMasterChef interface contains other functions like `exit` or `totalSupply` which are not present within the Qi DAO Staking Pool.
Resolution	RESOLVED The client has moved to the `pending` function and has removed the uncommon functions.

Issue #27	**Fee mechanism does not support transfer tax rewards in case the `feeTokenAddr` is set to a different token**
Severity	LOW SEVERITY
Location	Line 357 `_swapUniswapWithPath(rewardToFeeTokenPath, _feeAmount, feeTokenRouterAddr);`
Description	The `collectFee` function will do a basic swap which does not support transfer taxes if the `feeTokenAddr` is not set to the `rewardTokenAddr`. This functionality will break if the reward token has a transfer tax, causing harvests to revert.
Recommendation	Consider using `_swapUniswapWithPathForFeeOnTransferTokens` instead. Note that this update should also be made in the `harvest()` function.
Resolution	RESOLVED The recommended function has been used instead.

Issue #28	**Uniswap routing paths might be inefficient**
Severity	INFORMATIONAL
Description	Currently the harvest swaps are routed through specific pathways which might not be optimal for all assets in case this strategy is forked.
Recommendation	Consider carefully redefining the routing paths (and routers) whenever the strategy is changed.
Resolution	RESOLVED The client has confirmed they will carefully redefine these routes with each strategy.

Issue #29	Error message still references “Titan”
Severity	INFORMATIONAL
Location	Line 343 `require(newCutoff <= 10**18, "New cutoff must be less than 1 Titan");`
Description	The StrategyFarmTwoAssets contract still references a Titan token.
Recommendation	Consider adjusting the reference.
Resolution	RESOLVED

Issue #30	`wantToken` should never be set to the `feeTokenAddr`
Severity	INFORMATIONAL
Description	The `wantToken` should never be equal to the `feeTokenAddr`. If it would be set equal to said token address, the fee code might take out want tokens.
Recommendation	Consider adding this as an explicit requirement in the constructor to prevent potential issues when the code is forked.
Resolution	RESOLVED

Issue #31	Token symbol exceeds 11 characters which makes adding it to MetaMask more cumbersome
Severity	INFORMATIONAL
Description	Although the ERC-20 metadata standard does not specify a maximum length for a token symbol, MetaMask does not allow the length to exceed 11 characters. Adding any token to MetaMask with a symbol that is over 11 characters will require the user to manually adjust the symbol, which could be considered bad UX. In this case the `pairName` is used to generate the Jar share token symbol by putting a v before it, only 10 characters should be used. Furthermore, the `name` and `pairName` seem irrelevant to the actual strategy name and pair.
Recommendation	Consider whether it is possible to remove a single letter from the symbol string to make it compliant with MetaMask without user intervention.
Resolution	ACKNOWLEDGED


PALADIN
BLOCKCHAIN SECURITY