PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Meso Finance

11 October 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 Meso Finance on the Fantom Opera 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	Meso Finance
URL	https://www.meso.finance/
Platform	Fantom Opera
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`MesoVault`	`MesoVault.sol`	PENDING
`StratManager`	`StratManager.sol`	PENDING
`FeeManager`	`FeeManager.sol`	PENDING
`BaseMesoStrategyLP`	`BaseMesoStrategyLP.sol`	PENDING
`Source`	https://github.com/MesoFinance/ftm-vault-contracts `Resolution commit hash: fcc68b3cd5912d7980fe93318e8595ea9b52428c`	

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	6	6	-	-
Medium	5	5	-	-
Low	7	6	-	1
Informational	11	8	-	3
Total	**29**	**25**	**-**	**4**

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 MesoVault

ID	Severity	Summary	Status
01	HIGH	Deposit fee functionality is broken and will block all deposits	RESOLVED
02	HIGH	Upgrades to a malicious strategy allow the dev to withdraw all staked funds after the timelock delay of proposing this malicious upgrade has expired	RESOLVED
03	HIGH	Lack of validation of old and new `want` token when migrating strategies	RESOLVED
04	HIGH	In case the underlying Masterchef has deposit fees, governance could burn all funds by emergency withdrawing and calling `earn()` over and over again	RESOLVED
05	MEDIUM	Lack of lower limit validation for strategy's `approvalDelay`	RESOLVED
06	LOW	In case there are deposit fees, deposits can be prevented through an expensive attack by sending tokens to the vault in case there is a deposit fee or transfer tax	RESOLVED
07	LOW	Lack of validation that the investor received sufficient shares	RESOLVED
08	INFO	Lack of validation that the `earn` function actually increases the vault value	RESOLVED
09	INFO	Tokenomics: Deposits are inefficient for tokens with transfer taxes	ACKNOWLEDGED
10	INFO	Lack of events for `deposit`, `withdraw` and `incaseTokensGetStuck`	RESOLVED
11	INFO	Lack of check for receipt token if destination is vault address	ACKNOWLEDGED
12	INFO	Gas optimization: `withdraw` function could be simplified	ACKNOWLEDGED

1.4.2 StratManager

ID	Severity	Summary	Status
13	MEDIUM	Gov privilege: Swap router can be changed to steal rewards and dust LP tokens	RESOLVED
14	INFO	Lack of events for `setHarvester`, `setKeeper`, `setStrategist`, `setUnirouter`, `setVault` and `setFeeRecipient`	RESOLVED

1.4.3 FeeManager

ID	Severity	Summary	Status
15	LOW	`setFee` is defined in the strategy and not in the FeeManager	RESOLVED

1.4.4 BaseMesoStrategyLP

ID	Severity	Summary	Status
16	HIGH	Contracts are undefined on Fantom	RESOLVED
17	HIGH	outputToLp0Route and outputToLp1Route are wrongly defined	RESOLVED
18	MEDIUM	Unecessary owner failover might be marked as rug code by third-party reviewers	RESOLVED
19	MEDIUM	Lack of constructor parameter validation could lead to loss of funds if misconfigured	RESOLVED
20	MEDIUM	Protocol is incompatible with USDC staking and earning tokens	RESOLVED
21	LOW	Governance privilege: Calling `panic` and `deposit` iteratively will result in all funds being sent to the Masterchef deposit `feeAddress` in case there would be deposit fees	RESOLVED
22	LOW	`setFee` does not allow setting the fee to the maximum	RESOLVED
23	LOW	Dust tokens will accumulate over time	RESOLVED
24	LOW	Withdrawals might fail in case the token supply is extremely large	ACKNOWLEDGED
25	INFO	`input`, `output`, `output2`, `usdc`, `wmatic` and `masterchef` can be marked as `constant`	RESOLVED
26	INFO	Swaps do not support tokens with transfer taxes	RESOLVED
27	INFO	`retireStrat` can be removed if upgradeability is removed	RESOLVED
28	INFO	Lack of events for `setFee`, `managerHarvest`, `setHarvestOnDeposit`, `panic` and `retireStrat`	RESOLVED
29	INFO	`panic` can be made `external`	RESOLVED

2 Findings

2.1 MesoVault

The MesoVault is a straightforward fork of Beefy Finance, with the main change being a custom deposit fee.

When a deposit is done, the vault will transfer the underlying token to the strategy and mint a receipt token that represents the amount of shares the user has in the vault. Similarly, when a withdrawal is done, the receipt token is burnt and the amount of tokens equal to the shares burnt will be transferred to the user.

The vault's strategy is set on contract deployment, and can be changed by the owner after a delay based on the value of delay set during deployment. This delay value cannot be changed.

Although it is expected that ERC20 compliant LP tokens are used as the `want` token, in the case where tokens with a transfer fee are used as the `want` token, the contract supports this by checking the token balance of the vault before and after the `transferFrom` call, and using the difference as the actual deposit amount.

A fee initially set to 0% (can be adjusted to a maximum of 5%) is levied on deposits and sent to 0x2569c0423B69Ab70250D5B2Fc66803195Cb54AcD, a gnosis safe.

It should be noted that Paladin only audited the strategies provided (StrategyDfynRouterLP) and not any other strategy the protocol might add in the future. Investors should exercise extra caution when interacting with strategies that were not audited.

2.1.1 Privileged Roles and Actions

The following functions can be called by the owner of the contract:

- `proposeStrat`
- `upgradeStrat`
- `inCaseTokensGetStuck`

2.1.2 Issues & Recommendations

Issue #01	Deposit fee functionality is broken and will block all deposits
Severity	HIGH SEVERITY

Location	Lines 1155-1159
	`function deposit(uint _amount) public nonReentrant {`
	`  strategy.beforeDeposit();`
	`  uint256 beforeBalance = input().balanceOf(msg.sender);`
	`  uint256 withdrawalFeeAmount = _amount.mul(10000).div(10000000);`
	`  IERC20(input()).safeTransfer(gnosisWallet,`
	`beforeBalance.sub(withdrawalFeeAmount));`

Description	The code contains functionality to send a 0.1% deposit fee to the gnosis safe of the governance. However, this code is badly implemented in that it tries to transfer tokens from the vault to the multisig before any deposit has occurred. As there are unlikely any tokens in the vault, this will cause deposits to revert. Furthermore, it tries to transfer 99.9% of the user balance of the token to the vault. However, since this code is badly implemented (it still uses the vault as source for the 99.9%, and not the user wallet) this potential rug is not possible and deposits revert.

Recommendation	Consider properly handling the deposit fee as is described in the example below.
	
	`uint256 depositFee = _amount.div(1000);`
	`uint256 _pool = balance();`
	`input().safeTransferFrom(msg.sender, gnosisWallet, depositFee);`
	`input().safeTransferFrom(msg.sender, address(this),`
	`_amount.sub(depositFee));`
	`earn();`
	`uint256 _after = balance();`

Resolution	RESOLVED The recommended fix has been added.

Issue #02	**Upgrades to a malicious strategy allow the dev to withdraw all staked funds after the timelock delay of proposing this malicious upgrade has expired**
Severity	HIGH SEVERITY
Location	Line 1235 `function upgradeStrat() public onlyOwner {`
Description	As the Meso vaults are forked from Beefy Finance, it contains the same upgradeability code as Beefy has. The idea is that when the vault strategy changes over time, users do not need to restake, instead the developers can simply upgrade the code to a new strategy. If a malicious contract is chosen to upgrade to by the governance, this allows the governance to steal all staked funds. However, for some of the vaults, this might be seen as an excessive privilege. Among others, it is very unlikely that the SpookySwap staking contract is going to change or break for example. For these simple strategies, it is likely worth it to remove this governance privilege in favour of promoting decentralisation and investor confidence that Meso cannot run with the users' their money. Finally, on simple strategies like compounding AMM rewards, third-party reviewers like RugDoc may mark this privilege as excessive and either mark the vault as high-risk or non-eligible. The main risk in our experience with simple staking contracts is that the withdraw function could break due to the reward mechanism. However, this issue is already taken care of in the panic() method in the strategies, which withdraws all funds in emergencies without interacting with the reward mechanism. It should be noted that this is exactly the same upgradeability code as Beefy has and Beefy has similar governance privileges. It should also be noted that larger investors can protect themselves by actively listening to NewStratCandidate events emitted by the vault. These events announce that an upgrade can happen after the approval delay expires. These investors can then review the new strategy and decide to unstake if it is malicious.

Recommendation	Consider whether upgradeability is a necessary requirement. The client could consider making a poll for their users to see which option they prefer. If the client is comfortable with asking investors to restake in a new vault when a strategy adjustment has to be made, `upgradeStrat`, `proposeStrat` and `StratCandidate` functions can be removed.
Resolution	RESOLVED All upgrade functionality has been removed.

Issue #03	Lack of validation of old and new want token when migrating strategies
Severity	HIGH SEVERITY
Location	Line 1219 `function proposeStrat(address _implementation) public onlyOwner {`
Description	While there is a validation of the new strategy in `proposeStrat` to ensure that the vault address of it is the same as the vault itself, there is no guarantee that the input token of the old and new strategy is the same. If the input token is different (e.g Token A for old strategy, B for new strategy) and a migration goes through, the old input tokens (A) would be transferred from the old strategy into the vault, and the new `want` token (B) would be deposited to the new strategy. In such a case, users will be unable to withdraw their original deposited tokens as the input token has changed. Also, since the input token has now changed, the original input tokens can be withdrawn by the owner using `inCaseTokensGetStuck`.

Recommendation	Consider adding a check in proposeStrategy that ensures that the want token hasn't changed compared to the current strategy.
	
	Lines 1219-1227
	`function proposeStrat(address _implementation) public onlyOwner {`
	`    require(address(this) == IStrategy(_implementation).vault(), "Proposal not valid for this Vault");`
	`    require(input() == IStrategy(_implementation).input(), "Different input");`
	`    stratCandidate = StratCandidate({`
	`        implementation: _implementation,`
	`        proposedTime: block.timestamp`
	`    });`
	
	`    emit NewStratCandidate(_implementation);`
	`}`

Resolution	RESOLVED `proposeStrat` has been removed.

Issue #04	**In case the underlying Masterchef has deposit fees, governance could burn all funds by emergency withdrawing and calling `earn()` over and over again**
Severity	HIGH SEVERITY
Description	A lot of the common Masterchefs have or allow for deposit fees on their pools. If the governance of this vault ever turns truly malicious, they could repeatedly call the `panic` and `earn` methods over and over again until all funds are lost to the deposit fees (or transfer taxes).
Recommendation	Consider allowing emergency withdrawal functions (`panic` and `retireStrat`) to only be called once on all underlying strategies, closing the strategy permanently and preventing unpausing.
Resolution	RESOLVED After `panic` has been called, depositing and harvesting becomes impossible due to a `panicStatus` flag by the strategy.

Issue #05	**Lack of lower limit validation for strategy's `approvalDelay`**
Severity	MEDIUM SEVERITY
Description	Although the `approvalDelay` cannot be modified after initialization in the constructor, it is possible to set a value of 0, or a low value. This can allow instant or almost instant changes to the underlying strategy. The severity for this is adjusted as it is set in the constructor, so users can verify the state variable which cannot be modified before depositing into the vault.
Recommendation	Consider adding a lower limit check for `approvalDelay` in the constructor. For example, if the lower limit is 7 days, the value should be >= 7 days. `require(_approvalDelay >= 7 days, "Insufficient approval delay");`
Resolution	RESOLVED The upgrade functionality has been removed completely.

Issue #06	**In case there are deposit fees, deposits can be prevented through an expensive attack by sending tokens to the vault in case there is a deposit fee or transfer tax**
Severity	LOW SEVERITY

Location	Line 1155-1173
	`function deposit(uint _amount) public nonReentrant {`
	`strategy.beforeDeposit();`
	`...`
	`uint256 _pool = balance();`
	`...`
	`earn();`
	`uint256 after = balance();`
	`_amount = _after.sub(_pool); // Additional check for deflationary tokens`
	`...`
	`}`

Description	To allow deposits to be made in Masterchefs with deposit fees, the vault checks the balance of the vault before and after the deposit and adds the incrementation to the user. However, a malicious actor could transfer in tokens in the vaults which would then be staked as well in the `earn()` call. Since these tokens are already added to `balance()`, the only impact they have on the final `_amount` is a negative one, since the vault value decreases as they are staked. Exploit specification: 1. A user wishes to deposit 1 USDC in a vault that deposits this in a 4% deposit fee Masterchef. 2. Before the user does this, a malicious attacker transfers in 30 USDC to the vault (using a simple ERC20 transfer). 3. When the user deposits, the vault actually deposits 31 USDC in the Masterchef resulting in a 1.24 USDC fee. 4. Since the vault value after the deposit is larger than before it, the subtraction is negative and reverts the deposit.

Recommendation	In case the underlying strategies do not have any deposit fees, no changes need to be made, except for the case where tokens with transfer taxes will ever be added. In case strategies with deposit fees are added, this potential vector needs to be considered.
	
	In case the client wishes to resolve this issue, an amount parameter could be added to the internalized `earn()` function to only deposit the deposited amount.
	
	`function earn(uint256 amount) internal {`
	`  want().safeTransfer(address(strategy), amount);`
	`  strategy.deposit(amount);`
	`}`

Resolution	RESOLVED The recommended code has been added.

Issue #07	**Lack of validation that the investor received sufficient shares**
Severity	LOW SEVERITY
Description	There is currently no validation in the `deposit` function to verify that the user received sufficient shares. An example where this could be problematic is if the underlying strategy deposits in a deposit-fee based masterchef and the fees are set to 100%. In this case the user receives zero shares but their funds will still be deposited.
Recommendation	Consider adding a check in the `deposit` or `earn` function to ensure that a fraction (eg. 90%) of the balance/deposit amount has been added to the `balance()`.
Resolution	RESOLVED The client has added in a check that tests if the underlying farm has a deposit fee of over 20%.

Issue #08	Lack of validation that the earn function actually increases the vault value
Severity	INFORMATIONAL
Description	The `earn` function is used to compound the vault. However, in case this compound actually leads to a loss in the underlying strategy (for example due to tokens with transfer taxes after an `emergencyWithdraw` is done), this could lead to all depositors having a reduced value per share.

Recommendation	Consider reverting harvest calls that reduce the share value. However, since `earn()` is used to deposit tokens with transfer taxes and deposit in Masterchefs with a deposit fee, this will prevent all deposits. Thus the client should consider making the current `earn()` implementation internal and only adding the requirement to a new external `harvest()` function.
	
	`function harvest() external {`
	`  uint256 _prevBal = balance();`
	`  earn();`
	`  require(balance() >= _prevBal, "not profitable");`
	`}`
	
	`function earn() internal {`
	`  uint256 _bal = available();`
	`  want().safeTransfer(address(strategy), _bal);`
	`  strategy.deposit();`
	`}`
	
	A similar check could be added to the `deposit` and `harvest` function to ensure that the value of shares is not reduced there either, but in that case the balance/share should be verified since both numbers will usually increase.
	
	Note that this harvest() functionality now suffers from a similar denial of service vector as the deposit() function. This could be prevented by adding an explicit amount parameter to harvest.

Resolution	RESOLVED The recommended check has been added.

Issue #09	Tokenomics: Deposits are inefficient for tokens with transfer taxes
Severity	INFORMATIONAL
Description	Since deposits first transfer the funds in the vault then to the strategy and finally into the actual underlying protocol, this might cause a significant loss of funds compared to direct staking.
Recommendation	Consider the implications of this if the vault is ever considered for tokens with transfer taxes.
Resolution	ACKNOWLEDGED

Issue #10	Lack of events for `deposit`, `withdraw` and `incaseTokensGetStuck`
Severity	INFORMATIONAL
Description	The above functions do not emit any events, even though such functions change the state of the contract.
Recommendation	Add events for the above functions.
Resolution	RESOLVED

Issue #11	Lack of check for receipt token if destination is vault address
Severity	INFORMATIONAL
Description	Users might mistakenly send the receipt token to the vault thinking that it would allow them to redeem their underlying, and lose funds in the process. In such a case, the vault owner would be able to recover those tokens using the `inCaseTokensGetStuck` function, but there is no use case where the receipt token is required to be sent to the vault. Note that although this strictly speaking is not compliant with the ERC-20 specification, many users have sent Moo tokens to beefy vaults directly, losing them permanently.
Recommendation	Consider modifying the transfer function to revert if the destination address is the vault contract address. `function transfer(address recipient, uint256 amount) public override returns (bool) { require(recipient != address(this), "!Use deposit function"); return super.transfer(); }`
Resolution	ACKNOWLEDGED

Issue #12	Gas optimization: `withdraw` function could be simplified
Severity	INFORMATIONAL
Description	In the `withdraw` function, a before-after pattern is done to calculate how much tokens are withdrawn from the strategy. Then `r`, the amount to be sent to the user, is set to `b.add(_diff)`. This could be simplified to save gas.

Recommendation	The following code in `withdraw` can be simplified as such:
	
	As is:
	`if (b < r) { uint _withdraw = r.sub(b); strategy.withdraw(_withdraw); uint _after = want().balanceOf(address(this)); uint _diff = _after.sub(b); if (_diff < _withdraw) { r = b.add(_diff); } }`
	
	Recommended:
	`if (b < r) { uint _withdraw = r.sub(b); strategy.withdraw(_withdraw); uint _after = want().balanceOf(address(this)); if(_diff < _withdraw) { r = _after; } }`

Resolution	ACKNOWLEDGED

2.2 StratManager

The StratManager is a dependency implemented by the strategies. It stores important governance-related variables and controls.

2.2.1 Privileged Roles and actions

The following functions can be called by the owner of the contract:

- `setHarvester`
- `setKeeper`
- `setStrategist`
- `setUnirouter`
- `setVault`

2.2.2 Issues & Recommendations

Issue #13	**Gov privilege: Swap router can be changed to steal rewards and dust LP tokens**
Severity	MEDIUM SEVERITY
Location	Line 1327 `function setUnirouter(address _unirouter) external onlyOwner {`
Description	The owner can change the router which is used to convert rewards into lp tokens, setting this router to a malicious one can be abused to leach the rewards. ❗ A swappable router was the cause of the recent AFK rugpull - having this inside the protocol might put off investors who were affected by this rug.
Recommendation	Consider whether the `uniRouter` ever needs to change and whether a redeployment is not easier at that time. If the `uniRouter` should remain changeable, consider putting the strategies behind a sufficiently long timelock.
Resolution	RESOLVED This privilege has been removed.

Issue #14	**Lack of events for `setHarvester`, `setKeeper`, `setStrategist`, `setUnirouter`, `setVault` and `setFeeRecipient`**
Severity	INFORMATIONAL
Description	Functions that affect sensitive variables should emit events as notifications.
Recommendation	Add events to the above functions.
Resolution	RESOLVED

2.3 FeeManager

The FeeManager is a dependency implemented by the various strategies. It keeps track of the strategist fee which is levied on every harvest. This fee is initially set to 5% and can be set up to 6%.

2.3.1 Issues & Recommendations

Issue #15	`setFee` is defined in the strategy and not in the FeeManager
Severity	LOW SEVERITY
Description	The strategy itself contains the function to adjust the strategist fee; however, this does away with the need of the FeeManager as the whole point of it is to manage the fees.
Recommendation	Consider moving the `setFee` function to the FeeManager dependency and renaming it to `setStrategistFee` to avoid ambiguity. ❗ Don't forget to also add an event to this function when it is moved.
Resolution	RESOLVED The function has been moved to the FeeManager.

2.4 BaseMesoStrategyLP

The BaseMesoStrategyLP is a strategy used by the protocol. It stakes and compounds tokens in the underlying Masterchef.

2.4.1 Issues & Recommendations

Issue #16	Contracts are undefined on Fantom
Severity	HIGH SEVERITY
Description	The protocol uses constant contracts (router, Masterchef, tokens, etc.) which do not exist on Fantom. Instead, these are only defined on Polygon.
Recommendation	Consider using contracts which actually exist on the deployment chain.
Resolution	RESOLVED The contract has been updated to use the Meso Masterchef.

Issue #17	`outputToLp0Route` and `outputToLp1Route` are wrongly defined
Severity	HIGH SEVERITY

Location	Lines 1323-1329
	`outputToLp0Route = new address[](2);`
	`outputToLp0Route[0]= output;`
	`outputToLp0Route[1]= output2;`
	
	`outputToLp1Route = new address[](2);`
	`outputToLp1Route[0]= output;`
	`outputToLp1Route[1]= output2;`

Description	The two routes to swap the output to lp token 0, and lp token 1 wrongly swaps the tokens to `output2`.
Recommendation	Consider adding `lpToken0` and `lpToken1` as the destinations of the two routes.
Resolution	RESOLVED These routes have been updated to route to the lp subtokens.

Issue #18	**Unecessary owner failover might be marked as rug code by third-party reviewers**
Severity	MEDIUM SEVERITY

Location	Lines 1367-1371
	`if (tx.origin == owner() || paused()) {`
	`  IERC20(input).safeTransfer(vault, wantBal);`
	`} else {`
	`  IERC20(input).safeTransfer(vault, wantBal);`
	`}`

Description	The `withdraw` function contains two identical code-paths, however, since the first one is executed only by the owner, this might cause third-party auditors into thinking that the owner can do a special withdrawal. It should be noted that both execution paths are identical but this issue is marked as medium risk since writing the code like this will likely have a negative impact on the risk assessment by third-party reviewers.
Recommendation	Consider rewriting the code by removing the if/else completely: `IERC20(input).safeTransfer(vault, wantBal);`
Resolution	RESOLVED The recommended simplification has been implemented.

Issue #19	**Lack of constructor parameter validation could lead to loss of funds if misconfigured**
Severity	MEDIUM SEVERITY
Description	Under certain circumstances, the UniSwap operations might remove `want` tokens since there is an overlap between the configured tokens.
Recommendation	Consider adding validation to the constructor to ensure that the input token is not equal to the output token or one of the LP tokens. Furthermore in case the following issue is not implemented, the constructor should validate that the input and output token are both not equal to USDC.
Resolution	RESOLVED Validation was added, and although there was no validation for the USDC token, the recommendation of the following issue makes this protocol compatible with USDC.

Issue #20	**Protocol is incompatible with USDC staking and earning tokens**
Severity	MEDIUM SEVERITY
Description	The `chargeFees` function sends the whole USDC balance to the strategist, therefore if the staking (input) or earning (output) token is ever set to USDC, the whole stake or harvest will be sent to the strategist. This first case would occur in case `panic` is called.
Recommendation	Consider setting the to variable on the swap directly to the strategist address and removing the explicit transfer. `IUniswapRouterETH(unirouter).swapExactTokensForTokens(toUsdc,0, outputToUsdcRoute, strategist, now);`
Resolution	RESOLVED The swap happens directly to the strategist now.

Issue #21	**Governance privilege: Calling `panic` and `deposit` iteratively will result in all funds being sent to the Masterchef deposit `feeAddress` in case there would be deposit fees**
Severity	LOW SEVERITY
Description	If there would be a deposit fee, iteratively unstaking and staking the vault will result in the vault value declining over time. A malicious governance could use this to drain the vault to the Masterchef `feeAddress`.
Recommendation	Consider removing `retireStrat` (or making it non-upgradeable) and only making `panic` callable once through a `panicked` boolean state variable.
Resolution	RESOLVED `panic` can only be called once now.

Issue #22	**`setFee` does not allow setting the fee to the maximum**
Severity	LOW SEVERITY
Location	Line 1340 `require(_fee < MAX_FEE,"Must be less than MAX_FEE");`
Description	`setFee` checks that `_fee` is less than `MAX_FEE`, so a value equal to `MAX_FEE` cannot be set.
Recommendation	Consider using greater than or equal to. `require(_fee <= MAX_FEE,"Must be less than MAX_FEE");`
Resolution	RESOLVED

Issue #23	Dust tokens will accumulate over time
Severity	LOW SEVERITY
Description	Due to imbalances in the swaps, either `lpToken0` or `lpToken1` will slowly accumulate in the strategy over time. This will always only be a fraction of the value so this issue is not that severe.
Recommendation	Consider adding a function that converts `lpToken0` and `lpToken1` back to the `want` token.
Resolution	RESOLVED A governance function to take these out has been added.

Issue #24	Withdrawals might fail in case the token supply is extremely large
Severity	LOW SEVERITY
Description	Within withdrawal, a multiplication is done which could at most go to `totalSupply`^2 for a token. In case this multiplication overflows, withdrawals would become impossible. This issue is marked as informational since we know of no tokens with such a high total supply.
Recommendation	Consider this carefully when picking which tokens are added.
Resolution	ACKNOWLEDGED

Issue #25	`input`, `output`, `output2`, `usdc`, `wmatic` and `masterchef` can be marked as `constant`
Severity	INFORMATIONAL
Description	Variables that are defined within the contract and are never modified can be marked as `constant` to save gas.
Recommendation	Consider marking the above variables as `constant`.
Resolution	RESOLVED

Issue #26	Swaps do not support tokens with transfer taxes
Severity	INFORMATIONAL
Description	The swapping logic in `chargeFees` and `addLiquidity` does not support tokens with transfer taxes.
Recommendation	Consider using `swapExactTokensForTokensSupportingFeeOnTransferTokens` instead of `swapExactTokensForTokens`.
Resolution	RESOLVED

Issue #27	`retireStrat` can be removed if upgradeability is removed
Severity	INFORMATIONAL
Description	In case the client decides to remove upgradeability, `retireStrat` can be removed since it can only be called by the vault.
Recommendation	Consider removing `retireStrat` if upgradeability is removed.
Resolution	RESOLVED

Issue #28	**Lack of events for `setFee`, `managerHarvest`, `setHarvestOnDeposit`, `panic` and `retireStrat`**
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Consider adding events to the above functions.
Resolution	RESOLVED

Issue #29	**`panic` can be made `external`**
Severity	INFORMATIONAL
Description	The panic function 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 this function `external`.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY