PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For HoneyFarm (HoneyMoon)

04 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 HoneyMoon, HoneyFarm's Layer 3 on the Binance Smart Chain (BSC). 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	HoneyMoon
URL	http://moon.honeyfarm.finance
Platform	Binance Smart Chain
Language	Solidity

1.2 Contracts Assessed

The client has provided us with HoneyFarm's contracts on Github to be audited, though their deployed contracts are for their latest layer, HoneyMoon. It is to be noted that HoneyFarm and HoneyMoon are similar, with the only changes in the latest layer being that transfer tax is set to 0 currently in the token.

Name	Contract	Live Code Match
YetiMaster	0x671e56C68047029F236f342b18632425C75885a3	MATCH
HoneyReferral	0x9b17fe2cE39b1C5BcB6BC7A60781ECdC9adb6032	MATCH
HoneyToken	0xE8c93310af068aa50bd7bF0ebFa459Df2a02ceba	MATCH
StrategyChef	0xBDE0b81ABd4156705a26829Ecd9789187CA0D35d	MATCH

The client has deployed some of their contracts with unconventional names. YetiMaster is the VaultChef of the farm and the HoneyToken is the MoonToken.

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2	0	1	1
Medium	8	5	-	3
Low	12	2	3	7
Informational	17	1	4	12
Total	**39**	**8**	**8**	**23**

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 YetiMaster

ID	Severity	Summary	Status
01	HIGH	`earningToken` ownership can be transferred to mint and dump	PARTIAL
02	MEDIUM	Users could be mislead into depositing in a malicious pool	RESOLVED
03	MEDIUM	`feeAddr` is not set in the constructor	RESOLVED
04	MEDIUM	`setWithdrawFee` does not have safeguards	RESOLVED
05	LOW	Duplicate strategies could be added	PARTIAL
06	LOW	`depositFeeBP` and `withdrawFeeBP` require `uint256` casting	PARTIAL
07	LOW	No token validation is done on strategy addition	ACKNOWLEDGED
08	LOW	`massUpdatePools` and `updatePool` will break if ever a broken strategy or non-strategy address is added as a pool	ACKNOWLEDGED
09	LOW	`updatePool` can block all deposits and withdrawals if `totalAllocPoint` is zero	ACKNOWLEDGED
10	LOW	Setting `feeAddr` and `devAddr` to the zero address will break most functionality	ACKNOWLEDGED
11	INFO	`earningToken`, `burnAddress`, `EarningsPerBlock`, `EarningsDevPerBlock`, `MAX_WITHDRAWAL_FEE_BP` and `MAX_DEPOSIT_FEE_BP` can be made `constant`	RESOLVED
12	INFO	Ensure that the hard-coded `earningToken` is correct	RESOLVED
13	INFO	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions	ACKNOWLEDGED
14	INFO	`msg.sender` is already an address and does not have to be cast to address again	ACKNOWLEDGED
15	INFO	There are no sanity checks in the `setEarningsReferral` function	ACKNOWLEDGED
16	INFO	`migrateToV2` event is unused	ACKNOWLEDGED
17	INFO	`add`, `set`, `inCaseTokensGetStuck`, `setDevAddress`, `setFeeAddress`, `updateEmissionRate`, `deposit`, `withdraw`, `emergencyWithdraw`, `setEarningsReferral`, `setReferralCommission`, `transferEarningTokenOwnership` and `setWithdrawFee` can be made `external`	ACKNOWLEDGED
18	INFO	`add`, `set`, `dev`, `setEarningsReferral`, `setReferralCommissionRate`, `transferEarningTokenOwnership`, `setWithdrawFee` should emit events	ACKNOWLEDGED

1.3.2 HoneyReferral

ID	Severity	Summary	Status
19	LOW	The owner of the Referral contract can overwrite themselves as the referrer for all users using the `recordReferral` function	ACKNOWLEDGED
20	INFO	`recordReferral`, `recordReferralCommission`, `getReferrer` can be made `external`	ACKNOWLEDGED
21	INFO	`drainBEP20Token` should emit an event	ACKNOWLEDGED

1.3.3 HoneyToken

ID	Severity	Summary	Status
22	HIGH	Operator can blacklist any address	ACKNOWLEDGED
23	LOW	`mint` function could have been used to pre-mint large amounts of tokens before ownership is transferred to HoneyFarm	RESOLVED
24	INFO	`burnRate` is redundant	RESOLVED
25	INFO	`setBlacklist`, `setTaxFreeList`, `transferOperator`, `updateTransferTaxRate` can be made `external`	ACKNOWLEDGED

1.3.4 StrategyChef

ID	Severity	Summary	Status
26	MEDIUM	Strategy could deposit into malicious Masterchef	RESOLVED
27	MEDIUM	`distributeFee` sends harvested tokens to the `feeAddress`	RESOLVED
28	MEDIUM	earned token can be changed	ACKNOWLEDGED
29	MEDIUM	`inCaseTokensGetStuck` is able to remove earned tokens	ACKNOWLEDGED
30	MEDIUM	Governance can stop tokens being deposited into underlying Masterchef	ACKNOWLEDGED
31	LOW	`deposit`, `withdraw`, `earn` and `distributeFee` are not strictly secure from reentrancy	PARTIAL
32	LOW	Lack of safeguards in `setFeeAddress`	ACKNOWLEDGED
33	LOW	`setPid` function is mis-specified	ACKNOWLEDGED
34	LOW	Unstaking transfer tax tokens may result in unintended side-effects.	RESOLVED
35	INFO	`buyBackAddress` is redundant	RESOLVED
36	INFO	`earnedToLazyMintPath` is redundant	RESOLVED
37	INFO	`deposit`, `withdraw`, `earn`, `pause`, `unpause`, `setGov`, `setFeeAddress`, `setEarnedAddress`, `setPid`, `setIsCompound` and `inCaseTokensGetStuck` can be made `external`	ACKNOWLEDGED
38	INFO	`deposit`, `withdraw`, `earn`, `pause`, `unpause`, `setGov`, `setFeeAddress`, `setEarnedAddress`, `setPid`, `setIsCompound` and `inCaseTokensGetStuck` should emit events	ACKNOWLEDGED
39	INFO	`isCAKEStaking`, `isCompound`, `farmContractAddress`, `pid`, `wantAddress`, `earnedAddress`, `YetiMasterAddress` can be made `immutable`	ACKNOWLEDGED

2 Findings

2.1 YetiMaster

This is a Vault Masterchef forked from AutoFarm with significant modifications. There are deposit and withdrawal fees, and unique strategies may be added as pools into which users deposit funds. The contract owner also has the ability to withdraw tokens sent here by mistake, but deposited tokens and earnings are safe in that they cannot be withdrawn by this function as deposited tokens are transferred directly to the strategy contract upon deposit.

2.1.1 Privileged Roles

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

- `add`
- `set`
- `inCaseTokensGetStuck`
- `setDevAddress`
- `setFeeAddress`
- `setEarningsReferral`
- `setReferralCommissionRate`
- `transferEarningTokenOwnership`
- `setWithdrawFee`

2.1.2 Issues & Recommendations

Issue #01	`earningToken` ownership can be transferred to mint and dump
Severity	HIGH SEVERITY
Description	`transferEarningTokenOwnership` can be called to transfer ownership of `earningToken` to any address, which can then mint and dump tokens at will. This would also cause `updatePool` to no longer mint tokens, and deposits and withdrawals will also revert.
Recommendation	Consider removing this function unless absolutely required. Alternatively, this can be partially resolved by setting ownership of the HoneyFarm contract to a reasonably long Timelock.
Resolution	PARTIALLY RESOLVED The client has stated that their CEO has undergone a RugDoc KYC, as well as a Korini Kindergarten KYC (Korean influencer, we are unable to verify this) which disincentivizes them from transferring ownership and minting (as there are serious repercussions to the disclosure of their identity). However, it should be noted that their RugDoc KYC was done on their previous layer, Honey, whilst this audit is for their next layer, Yeti. We therefore advise caution. Additionally, they have also stated that this is a necessary function of their layered farming, and will be required in events of migrating contracts.

Issue #02	Users could be mislead into depositing in a malicious pool
Severity	MEDIUM SEVERITY
Description	HoneyFarm pools are immutable, which means that the developer cannot change an existing pool to a new strategy. This is great from an investor's perspective. However, the developer can still add new pools using a token that already has an existing pool. In this case, the user will not have to reapprove the token and thus they may be unaware that they are depositing the token in a new pool. If the developer has dishonest intentions, this method could be used to trick investors by adding a new strategy that has malicious code.
Recommendation	Consider only using audited, transparent and safe strategy contracts and publishing the addresses of such verified contracts for each vault pool so that users may be able to ascertain that the strategy for which their funds will be deposited into is safe. Consider also adding a sufficiently long timelock to the farm so new pools can be vetted by the users.
Resolution	RESOLVED The client has acknowledged this issue, and will place the contract under a sufficiently long Timelock. Once that has been done, we will mark this issue as Resolved. They shall also avoid adding in malicious strategies due to their RugDoc KYC status.

Issue #03	`feeAddr` is not set in the constructor
Severity	MEDIUM SEVERITY
Description	As the `feeAddr` is not set in the constructor, it will be set to the zero address by default. If the protocol forgets to set this to a valid address using `setFeeAddress` at a later point, deposits, withdrawals and emergency withdrawals will revert as tokens cannot be transferred to the zero address.
Recommendation	Consider setting `feeAddr` in the constructor.
Resolution	RESOLVED

Issue #04	`setWithdrawFee` does not have safeguards
Severity	MEDIUM SEVERITY
Description	There are currently no safeguards in the `setWithdrawFee` function on the number of `withdrawFeeIntervals` and `withdrawalFeeBP` that can be added to the array. This essentially allows for an unreasonably large number of withdrawal fees and intervals to be set, and may force users to be charged withdrawal fees for any duration staked. Additionally, if the length of the arrays is too long, then the contract will run out of gas in the for loop.
Recommendation	Consider limiting the number of intervals and fees that be set in the arrays to a reasonable number, such as only having 3 withdrawal intervals and fees each.
Resolution	RESOLVED There is now a maximum of 2 withdrawal fees in the array.

Issue #05	Duplicate strategies could be added
Severity	LOW SEVERITY
Description	There are currently no checks to ensure that duplicate strategies are not added.

Recommendation	Consider adding an index of existing strategies to keep track of which ones are already added:
	`mapping(strategy => bool) public isExistingStrat;`
	In the add function the following code could be added:
	`require(!isExistingStrat[_strat]);`
	`isExistingStrat[_strat] = true;`

Resolution	PARTIALLY RESOLVED The client has stated that this is intended, and will take all necessary precautions to ensure that they will not add in duplicate strategies.

Issue #06	`depositFeeBP` and `withdrawFeeBP` require `uint256` casting
Severity	LOW SEVERITY
Description	As these functions utilize SafeMath which only supports `uint256`, the current `uint16` casting for `depositFeeBP` and `withdrawFeeBP` may not be compatible.
Recommendation	Consider explicitly casting `depositFeeBP` and `withdrawFeeBP` to `uint256` for SafeMath operations.
Resolution	PARTIALLY RESOLVED The client has stated that they will take the necessary precautions to ensure that the functions do not overflow and underflow, and that they believe `uint16` is more gas efficient.

Issue #07	No token validation is done on strategy addition
Severity	LOW SEVERITY
Description	When an EOA or non-ERC-20 contract is used as the want address of a pool by accident within the add function, it may result in an incompatible token being deposited into the strategy contract. There is also no checks to ensure that it is the same wantAddress specified in the strategy contract.
Recommendation	Consider adding in a require(want.totalSupply() > 0); check in the add function, as this ensures that the token contract is valid.
Resolution	ACKNOWLEDGED

Issue #08	`massUpdatePools` and `updatePool` will break if ever a broken strategy or non-strategy address is added as a pool
Severity	LOW SEVERITY
Location	Line 1394 `uint256 wantLockedTotal = IStrategy(pool.strat).wantLockedTotal();`
Description	If this call fails for any single pool, `massUpdatePools` will always fail as well. This behavior is then irrevertible as no pool can be removed. We've seen this accident happen in the past to other projects.
Recommendation	This can be resolved by acknowledging the issue and ensuring that the strategy contracts to be added are valid and non-malicious.
Resolution	ACKNOWLEDGED

Issue #09	**updatePool can block all deposits and withdrawals if `totalAllocPoint` is zero**
Severity	LOW SEVERITY
Description	If all `allocPoints` ever need to be set to zero to disable emissions, all `deposit` and `withdraw` functionality will be broken. This is because within the `updatePool` function, a `.div` call is made on the `totalAllocPoint` variable.

Recommendation	Change the safeguard check in the updatePool function from
	`if (block.number <= pool.lastRewardBlock) {`
	to
	`if (block.number <= pool.lastRewardBlock || totalAllocPoints == 0) {`

Resolution	ACKNOWLEDGED

Issue #10	**Setting `feeAddr` and `devAddr` to the zero address will break most functionality**
Severity	LOW SEVERITY
Description	Should the `feeAddr` be set to the zero address, transfers may fail for tokens that do not allow transfers to the zero address and thus break the `deposit` function. If the `devAddr` is set to the zero address then minting will fail, causing `_updatePool` to fail and thus deposits and withdrawals..

Recommendation	To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like
	
	`require(_feeAddress != address(0), "!nonzero");`
	`require(_devAddr != address(0), "!nonzero");`
	
	to the configuration function.

Resolution	ACKNOWLEDGED

Issue #11	**`earningToken`, `burnAddress`, `EarningsPerBlock`, `EarningsDevPerBlock`, `MAX_WITHDRAWAL_FEE_BP` and `MAX_DEPOSIT_FEE_BP` can be made `constant`**
Severity	INFORMATIONAL
Description	Variables that are only set in the constructor but never modified can be indicated as such with the `constant` keyword. This is considered best practice since it makes the code more accessible for third-party reviewers and potentially saves gas.
Recommendation	Consider making the above variables explicitly `constant`.
Resolution	RESOLVED

Issue #12	Ensure that the hard-coded `earningToken` is correct
Severity	INFORMATIONAL
Description	Currently, the earningToken address is hard-coded as 0x51C9242ebE6D982B4f572Be66d6fC2af7e0abe80. Assuming that the network to be used is BSCScan, this address currently points to an unverified tEar token contract.
Recommendation	No resolution is required. Kindly ensure that if that the `earningToken` address is correct upon deployment, and if `tEar` is truly to be used, to verify that token contract.
Resolution	RESOLVED

Issue #13	Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions
Severity	INFORMATIONAL

Description	Within `updatePool`, `accEarningsPerShare` is based upon the `wantLockedTotal` variable.
	
	`pool.accEarningsPerShare = pool.accEarningsPerShare.add(`
	`  earningsReward.mul(1e12).div(wantLockedTotal));`
	
	However, if this `wantLockedTotal` becomes a severely large value, precision errors due to rounding will occur. This is famously seen when protocols decide to add meme tokens which usually have huge supplies and no decimals.

Recommendation	Consider increasing precision to 1e18 across the entire contract.
Resolution	ACKNOWLEDGED

Issue #14	`msg.sender` is already an address and does not have to be cast to address again
Severity	INFORMATIONAL
Description	Throughout the codebase, `msg.sender` is explicitly cast to an address like in the following code: `address(msg.sender)`. Since `msg.sender` is already an address, this code is redundant (although it does no harm either).
Recommendation	Consider replacing all occurrences of `address(msg.sender)` with `msg.sender`.
Resolution	ACKNOWLEDGED

Issue #15	There are no sanity checks in the `setEarningsReferral` function
Severity	INFORMATIONAL
Description	A lot of functionality can break if the referral address is updated to a value that is not a referral contract. Furthermore, the referral contract could be upgraded by governance to a malicious one that sets themselves as the referral of everyone, allowing them to mint referral commissions to themselves.
Recommendation	Consider making the referral address non upgradeable (only settable once) to ensure that functionality can never break, such as setting it in the constructor. We rarely ever see a project updating their referral after it is initially set.
Resolution	ACKNOWLEDGED

Issue #16	`migrateToV2` event is unused
Severity	INFORMATIONAL
Description	This event does not look to be associated with any functions in the contract.
Recommendation	Consider removing this unused event.
Resolution	ACKNOWLEDGED

Issue #17	`add`, `set`, `inCaseTokensGetStuck`, `setDevAddress`, `setFeeAddress`, `updateEmissionRate`, `deposit`, `withdraw`, `emergencyWithdraw`, `setEarningsReferral`, `setReferralCommission`, `transferEarningTokenOwnership`, `setWithdrawFee` can be made `external`
Severity	INFORMATIONAL
Description	The above functions 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 these functions `external`.
Resolution	ACKNOWLEDGED

Issue #18	**`add`, `set`, `dev`, `setEarningsReferral`, `setReferralCommissionRate`, `transferEarningTokenOwnership`, `setWithdrawFee` should emit events**
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for these functions.
Resolution	ACKNOWLEDGED

2.2 HoneyReferral

The HoneyReferral contract is a storage contract to keep track of the referrals of users. It contains a `drainBEP20Token` function but this function can only take out tokens sent to the contract by accident and has no control over tokens in other parts of the system.

2.2.1 Privileged Roles

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

- `renounceOwnership`
- `transferOwnership`
- `updateOperator`
- `recordReferral`
- `drainBEP20Token`

2.2.2 Issues & Recommendations

Issue #19	**The owner of the Referral contract can overwrite themselves as the referrer for all users using the `recordReferral` function**
Severity	LOW SEVERITY
Description	The operator should only be the HoneyFarm contract. However, the owner of the Referral contract has privileges that may be abused. The following steps detail how the owner can make themselves the referrer for all users : 1. Owner of the Referral contract calls `updateOperator` to add themselves as an operator. 2. As an operator, they then call `recordReferral` with their own wallet address as the referrer for each user. 3. This results in their wallet address being minted potentially large amounts of referral commission.
Recommendation	There are 3 possible recommendations : 1. Consider making the `updateOperator` function callable only once. 2. Setting only the HoneyFarm as an operator in the modifier, and removing the `updateOperator` function. 3. Calling the `updateOperator` to set HoneyFarm as operator, then renouncing ownership of the Referral contract such that it cannot be called in the future.
Resolution	ACKNOWLEDGED

Issue #20	`recordReferral`, `recordReferralCommission`, `getReferrer` can be made `external`
Severity	INFORMATIONAL
Description	`recordReferral`, `recordReferralCommission`, `getReferrer` functions 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 these functions `external`.
Resolution	ACKNOWLEDGED

Issue #21	`drainBEP20Token` should emit an event
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add an event for this function.
Resolution	ACKNOWLEDGED

2.3 HoneyToken

This is an ERC20 token with transfer taxes that can be set to a maximum of 10%, and currently all of the transfer tax is slated to be burned. The token operator has the ability to exclude certain addresses from paying these transfer taxes, and also the ability to blacklist any addresses to disallow them from sending tokens.

2.3.1 Token Overview

Address	0xE8c93310af068aa50bd7bF0ebFa459Df2a02ceba
Token Supply	Unlimited
Decimal Places	18
Transfer Max Size	None
Transfer Min Size	None
Transfer Fees	Up to 10%
Pre-mints	5,500 pre-minted

2.3.2 Privileged Roles

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

- `setBlacklist`
- `setTaxFreeList`
- `transferOperator`
- `updateTransferTaxRate`
- `mint`

2.3.2 Issues & Recommendations

Issue #22	Operator can blacklist any address
Severity	HIGH SEVERITY
Description	`setBlacklist` function can be called by the operator to blacklist any address at any time, which essentially causes the address owner to be unable to send or sell tokens. This essentially makes the token become a honeypot to that address, and makes transferring any native tokens impossible. Note that this function should never be called on relevant operational contract such as the Honeyfarm, Router or Referral, for example.
Recommendation	Consider setting the Operator behind a very long Timelock, ideally 1 day or more, which removes the ability to arbitrarily call this function on any user. Consider also being transparent on when and who this function may be called. We realize that this function will probably be used to blacklist bots, though it may also unfortunately blacklist innocent addresses, including those that the operator does not fancy (if the operator has a vendetta against them for some reason, for example).
Resolution	ACKNOWLEDGED

Issue #23	`mint` function could have been used to pre-mint large amounts of tokens before ownership is transferred to HoneyFarm
Severity	LOW SEVERITY
Description	The `mint` function could be used to pre-mint tokens for legitimate uses including, but not limited to, the injection of initial liquidity, token presale, or airdrops; however, this function may also be used to pre-mint tokens for dumping.
Recommendation	Consider being forthright if this `mint` function has been used by letting your community know how much was minted, where they are currently stored, if a vesting contract was used for token unlocking, and finally the purpose of the mints.
Resolution	RESOLVED 5,500 tokens have been pre-minted for liquidity purposes, and token ownership has been transferred to the YetiMaster contract.

Issue #24	`burnRate` is redundant
Severity	INFORMATIONAL
Description	`burnRate` is not used in the contract, as the amount sent to the burn address is the taxed amount.
Recommendation	Consider removing `burnRate`.
Resolution	RESOLVED

Issue #25	`setBlacklist`, `setTaxFreeList`, `transferOperator`, `updateTransferTaxRate` can be made `external`
Severity	INFORMATIONAL
Description	The above functions 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 these functions `external`.
Resolution	ACKNOWLEDGED

2.4 StrategyChef

The StrategyChef is a simple strategy contract that primarily deposits into the Pancakeswap Masterchef or similar underlying Masterchef, and is owned by the HoneyFarm contract.

2.4.1 Privileged Roles

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

- `deposit`
- `withdraw`
- `pause`
- `unpause`
- `setGov`
- `setFeeAddress`
- `setEarnedAddress`
- `setIsCompound`
- `setPid`
- `inCaseTokensGetStuck`

2.4.2 Issues & Recommendations

Issue #26	Strategy could deposit into malicious Masterchef
Severity	MEDIUM SEVERITY
Description	Currently it is not specified in which underlying farm the strategy will deposit. This could be a malicious farm created by the developers that steals all tokens, or prevent withdrawals from being called.
Recommendation	Consider hard-coding the Masterchef that will be deposited into, and clearly notifying and displaying this to users so that they may be able to identify and verify the target Masterchef in use. Additionally, the protocol should ensure that the Masterchefs where funds will be deposited into should be non-malicious.
Resolution	RESOLVED The client has stated that they will only deposit into established protocols such as PancakeSwap, Qubit and Belt Finance, amongst others. They will also transparently document all contracts that will be used to their users, in addition to being disincentivized from depositing into malicious contracts due to their RugDoc KYC status.

Issue #27	`distributeFee` sends harvested tokens to the `feeAddress`
Severity	MEDIUM SEVERITY
Description	The comment for the function states that this will convert farm tokens into `want` tokens, though the function just sends the balance of earned tokens in the strategy contract to the `feeAddress`.
Recommendation	Consider if this is the intended behaviour of the function. Additionally, if the `feeAddress` is perhaps a redistribution contract, some clarification would indeed be well warranted. At its current state, we are of the opinion that this function sends harvest tokens to the `feeAddress` which may be an EOA.
Resolution	RESOLVED The client has stated that this is an intended function, and that the usage of the funds will be transparently documented to be used for buybacks, Royal Jelly, marketing and development fees.

Issue #28	`earned` token can be changed
Severity	MEDIUM SEVERITY
Description	The `setEarnedAddress` function currently gives the governance address the ability to change the address of earned tokens. We are unsure why there is this ability to change the strategy contract's earned token address.
Recommendation	Consider removing this function. If it is absolutely necessary, please clarify with us as we are concerned for the ability of governance to change the earned token address.
Resolution	ACKNOWLEDGED

Issue #29	`inCaseTokensGetStuck` is able to remove earned tokens
Severity	MEDIUM SEVERITY
Description	There is a safeguard against this function being called to take out `want` tokens, which is great. Unfortunately, the function is able to take out earned tokens, similar to the `distributeFee` function. We are unsure if this is intended behavior.
Recommendation	Consider adding in a safeguard against removing earned tokens, such as: `require(_token != earnedAddress, "!safe")`
Resolution	ACKNOWLEDGED

Issue #30	Governance can stop tokens being deposited into underlying Masterchef
Severity	MEDIUM SEVERITY
Description	The `setIsCompound` function sets the bool state for `isCompound`. If set to `true`, then tokens are deposited into the underlying target Masterchef such as Pancakeswap for farming. If set to `false`, then tokens are held in the strategy contract instead and sit idle rather than generate yield. We understand that this function may be used in the event that there is an issue in the underlying Masterchef, though that can be achieved using the `pause` function instead.
Recommendation	Consider removing this ability to change `isCompound`, and instead rely on the `pause` function should there be a need to stop deposits into the underlying Masterchef for any reason.
Resolution	ACKNOWLEDGED

Issue #31	**`deposit`, `withdraw`, `earn` and `distributeFee` are not strictly secure from reentrancy**
Severity	LOW SEVERITY
Description	Although there is a low risk of reentrancy here, we nevertheless highly recommend adding the `nonReentrant` modifier since tokens are deposited from HoneyFarm into the respective strategy contracts. This is to hedge against the edge case where ERC777 tokens may be added and used in these contracts.
Recommendation	Consider adding the `nonReentrant` modifier to all functions that deal with token transfers such as `deposit`, `withdraw`, `earn` and `distributeFee`.
Resolution	PARTIALLY RESOLVED The `withdraw` function has the nonReentrant modifier.

Issue #32	Lack of safeguards in `setFeeAddress`
Severity	LOW SEVERITY
Description	Setting `feeAddress` to the zero address may result in the `distributeFee` function failing as tokens cannot be sent to the zero address. The knock-on effect of this is that the `withdraw` and `earn` functions may also fail if `isCompound` is `true`.
Recommendation	Consider adding in a non-zero safeguard such as: `require(_feeAddress != address(0), "!nonzero");`
Resolution	ACKNOWLEDGED

Issue #33	`setPid` function is mis-specified
Severity	LOW SEVERITY
Description	The `setPid` function in its current form merely sets the bool state of `isCompound`. This is already achieved in the `setIsCompound` function. Additionally, should there be an ability to change the `pid` for which the strategy deposits into, it is a high risk issue as it allows for the governance to alter which pools are deposited into, including malicious contracts. The ability to thus change pids should not be present in the strategy contract.
Recommendation	Consider removing this function.
Resolution	ACKNOWLEDGED

Issue #34	Unstaking transfer tax tokens may result in unintended side-effects
Severity	LOW SEVERITY
Description	If transfer tax tokens are used, then the amount received in the strategy contract will be lesser than the amount sent from the underlying Masterchef contract that was staked when the withdraw function is called. To demonstrate, assume that token X has a 5% transfer tax. **Situation**: $100 of Token X is staked by the strategy into the underlying Masterchef. Due to transfer taxes, only $95 is ultimately staked. **Event**: `withdraw` is called to remove all staked Token X, therefore `_wantAmt` is $100. **Result**: Unfortunately, due to transfer taxes again, only $90.25 of Token X is received in the strategy contract. The `withdraw` function would then attempt to send $100 of Token X to the user, though the safeguards in Lines 1561-1567 should catch this issue.
Recommendation	The before-after method is implemented in the `deposit` function and in HoneyFarm, though not in `withdraw`. Unfortunately, implementing the before-after method in `withdraw` may incur unintended side-effects as there are several sensitive interactions between the various contracts involved. As such, this issue can resolved by acknowledging the issue and ensuring that if transfer tax tokens are to be used, then `isCompound` must be set to false such that tokens are held in the strategy contract and thus not subject to having its balance reduced when staking and unstaking in the underlying Masterchef.
Resolution	RESOLVED The client has stated that they will not support any transfer tax tokens in their strategies.

Issue #35	**buyBackAddress is redundant**
Severity	INFORMATIONAL
Description	buyBackAddress does not look to be used in the contract.
Recommendation	Consider removing buyBackAddress.
Resolution	RESOLVED

Issue #36	**earnedToLazyMintPath is redundant**
Severity	INFORMATIONAL
Description	earnedToLazymintPath does not look to be used as there is no conversion from earned tokens to the native Lazy token.
Recommendation	Consider removing this unused variable.
Resolution	RESOLVED

Issue #37	**`deposit`, `withdraw`, `earn`, `pause`, `unpause`, `setGov`, `setFeeAddress`, `setEarnedAddress`, `setPid`, `setIsCompound`, and `inCaseTokensGetStuck` can be made `external`**
Severity	INFORMATIONAL
Description	The above functions 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. Some of these functions are recommended to be removed, and as such labelling them as `external` would be redundant if they are removed.
Recommendation	Consider making these functions `external`.
Resolution	ACKNOWLEDGED

Issue #38	**`deposit`, `withdraw`, `earn`, `pause`, `unpause`, `setGov`, `setFeeAddress`, `setEarnedAddress`, `setPid`, `setIsCompound`, and `inCaseTokensGetStuck` should emit events**
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. Some of these functions are recommended to be removed, and as such emitting events for those functions is redundant if they are removed.
Recommendation	Add events for these functions.
Resolution	ACKNOWLEDGED

Issue #39	**`isCAKEStaking`, `isCompound`, `farmContractAddress`, `pid`, `wantAddress`, `earnedAddress` and `YetiMasterAddress` 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 and saves gas. Note that this also assumes that the ability to change `isCompound`, `pid`, `earnedAddress` is removed.
Recommendation	Consider making the above variables explicitly `immutable`.
Resolution	ACKNOWLEDGED


PALADIN
BLOCKCHAIN SECURITY