PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For IFOSwap

09 August 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 IFOSwap. 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	IFOSwap
URL	https://ifoswap.finance/
Platform	Binance Smart Chain
Language	Solidity

1.2 Contracts Assessed

The contracts below have been audited by Paladin. We advise users to check that the contract they are interacting with matches the one we have audited.

Name	Contract	Live Code Match
`Token`	`0x97641c20355571820F591839d972AD2d38ad9F00`	MATCH
`Masterchef`	`0xB9dbe638956a812650eb384958111B85B332AA24`	MATCH
`SmartChef`	`0xae75ca6f4b11c91df5339220f4db7d6f467751ed`	MATCH
`IFO`	`0xCF701a6809E30CF615eb0b446D62091a3Bf0cf0f`	MATCH
`Timelock`	`0xDceC0B7fcF2B842d56DEBb89521558bE2475665F`	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	5	3	1	1
Medium	8	2	1	5
Low	5	2	1	2
Informational	11	-	-	11
Total	**29**	**7**	**3**	**19**

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 Token

ID	Severity	Summary	Status
01	LOW	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef	ACKNOWLEDGED
02	INFORMATIONAL	`delegateBySig` can be frontrun and cause denial of service	ACKNOWLEDGED

1.3.2 Masterchef

ID	Severity	Summary	Status
03	HIGH	Migrator can be used maliciously to steal tokens	RESOLVED
04	HIGH	Deposit fees do not have an upper limit	RESOLVED
05	HIGH	Severely excessive rewards issue when a token with a transfer tax is added	PARTIALLY RESOLVED
06	MEDIUM	Duplicated pools may be added to the Masterchef	PARTIALLY RESOLVED
07	MEDIUM	`emergencyWithdraw` function is vulnerable to reentrancy	ACKNOWLEDGED
08	LOW	Minting will break if multiplier is set to a very high value	RESOLVED
09	LOW	Adding a EOA or non-token contract as a pool will break `updatePool` and `massUpdatePools`	RESOLVED
10	INFORMATIONAL	`dev` function can be renamed, made `external`, and only be called by the owner of the contract	ACKNOWLEDGED
11	INFORMATIONAL	Pools use the contract balance to figure out the total deposits	ACKNOWLEDGED
12	INFORMATIONAL	`deposit`, and `withdraw`, `emergencyWithdraw`, `add`, and `set` can be made `external`	ACKNOWLEDGED
13	INFORMATIONAL	`enterStaking` and `leaveStaking` are redundant	ACKNOWLEDGED
14	INFORMATIONAL	Lack of events for `add`, `set`, `setDevAddress` and `updateMultiplier`	ACKNOWLEDGED

1.3.3 SmartChef

ID	Severity	Summary	Status
15	MEDIUM	Severely excessive rewards issue when a token with a transfer tax is added	ACKNOWLEDGED
16	MEDIUM	Rewards can be stopped by owner at any time	ACKNOWLEDGED
17	MEDIUM	Withdrawing not possible in certain scenarios	ACKNOWLEDGED
18	MEDIUM	`emergencyWithdraw` function is vulnerable to reentrancy	ACKNOWLEDGED
19	INFORMATIONAL	Lack of event for `emergencyRewardWithdraw` and `stopRewards`	ACKNOWLEDGED
20	INFORMATIONAL	`emergencyRewardWithdraw`, `emergencyWithdraw`, `withdraw` and `deposit` can be made `external`	ACKNOWLEDGED
21	INFORMATIONAL	Constructor does not have parameter safeguards	ACKNOWLEDGED

1.3.4 IFO

ID	Severity	Summary	Status
22	HIGH	Admin can withdraw `lpTokens` meant for refunding after IFO ends, causing all funds to be stuck in the contract	RESOLVED
23	HIGH	User allocation is only precise up to 1/1000000	ACKNOWLEDGED
24	MEDIUM	No sanity checks in the constructor can lead to serious problems in the business logic	RESOLVED
25	MEDIUM	Deposit amounts will be miscalculated for tokens with a transfer tax before approve is called and will stop working once approve is called	RESOLVED
26	LOW	The IFO contract might stake in an insecure masterchef	ACKNOWLEDGED
27	LOW	Harvesting could be blocked in a number of cases	PARTIALLY RESOLVED
28	INFORMATIONAL	`lpToken`, `offeringToken`, `masterchef`, `pid` and `farmRewardToken` can be made immutable	ACKNOWLEDGED
29	INFORMATIONAL	Lack of events for `setOfferingAmount`, `setRaisingAmount`, `updateStartAndEndBlocks`, `approve`, `_endFarmStaking`, `finalWithdrawLPToken`	ACKNOWLEDGED

1.3.5 Timelock

No issues found.

2 Findings

2.1 Token

The contract allows for H2O tokens to be minted when the `mint` function is called by Owner, which is the contract deployer. This can be used to pre-mint tokens for various uses including injection of initial liquidity, token presale, airdrops, and others.

2.1.1 Token Overview

Address	0x97641c20355571820F591839d972AD2d38ad9F00
Token Supply	Unlimited
Decimal Places	18
Transfer Max Size	No maximum
Transfer Min Size	No minimum
Transfer Fees	None

2.1.2 Privileged Roles

The following `onlyOwner` functions can be called by the Masterchef:

- `mint`

2.1.3 Issues & Recommendations

Issue #01	`mint` function can be used to pre-mint large amounts of tokens before ownership is transferred to the Masterchef
Severity	LOW SEVERITY
Description	This could be used for the aforementioned reasons, or may be used to pre-mint and dump tokens when the token contract has been deployed but before ownership is set to the Masterchef contract. This risk is prevalent amongst less-reputable projects, and any pre-mints can be prominently seen on the blockchain.
Recommendation	Consider being forthright if this **`mint`** function is to be 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	ACKNOWLEDGED

Issue #02	`delegateBySig` can be frontrun and cause denial of service
Severity	INFORMATIONAL
Description	Currently if **`delegateBySig`** is executed twice, the second execution will be reverted. It is thus in theory possible for a bot to pick up **`delegateBySig`** transactions in the mempool and execute them before a contract can. The issue with this is that the rest of said contract functionality would be lost as well. This could be a problem in case it would have been executed by a contract that would have rewarded you for your delegation for example.
Recommendation	Consider adding the desired message sender in the structhash and requiring this desired sender to be equal to **`msg.sender`**. This reduces the problem to having the message sender be able to frontrun you which is okay if it is a reviewed contract.
Resolution	ACKNOWLEDGED

2.2 Masterchef

The Masterchef is an almost identical clone to PancakeSwap's Masterchef. The significant differences are the addition of deposit fees and the removal of the faulty syrup token.

2.2.1 Privileged Roles

The following `onlyOwner` functions can be called:

- `updateMultiplier`
- `add`
- `set`
- `setMigrator`

2.2.2 Issues & Recommendations

Issue #03	Migrator can be used maliciously to steal tokens
Severity	HIGH SEVERITY
Description	This function can be used to steal all staked assets in the Masterchef, and poses a significant risk of total loss to users.
Recommendation	Implementing a Timelock would not be sufficient as the risk of a honeytrap (ref: https://medium.com/goose-finance/the-honeytrap-a-story-about-a-migrator-behind-timelock-397b889ab780) will be ever present. Consider entirely removing the **`migrate`** function.
Resolution	RESOLVED A SafeOwner wrapper contract (`0x20db9346a4a385aaafa274a3a3cbea4e4d324cd3`) was implemented as the owner of the Masterchef, and the `migrate` function is no longer callable. We have also confirmed that the `migrate` function was not called prior to ownership being transferred to the SafeOwner contract (this would be known as the Honeytrap situation).

Issue #04	Deposit fees do not have an upper limit
Severity	HIGH SEVERITY
Description	Deposit fees can be set to any amount up to 100%, which can result in significant or total loss of funds on unsuspecting users.

Recommendation	Consider setting a limit to a more reasonable amount, such as 4% (currently the norm) or up to 10%, in both `add` and `set` functions.
	
	For **`add`**:
	
	`require(_depositFeeBP <= 400, "add: invalid deposit fee basis points");`
	
	For **`set`**:
	
	`require(_depositFeeBP <= 400, "set: invalid deposit fee basis points");`

Resolution	RESOLVED Capped to 4% in the SafeOwner contract.

Issue #05	Severely excessive rewards issue when a token with a transfer tax is added
Severity	HIGH SEVERITY
Description	When tokens with a transfer tax are added to the pools, it will result in significant excessive rewards due to the way the Masterchef handles the reward mechanism: rewards can be heavily inflated when the balance of the Masterchef no longer matches that of user deposits. This happens for example with transfer tax tokens. This flaw of the Masterchef has recently been exploited on a significant number of projects, all of which had their native tokens go to $0 after the exploit. This issue was also present in Sushiswap (the original Masterchef). Since they were never meant to have any tokens but LP tokens, it was not a problem for them, but has become a problem to projects who have started forking it for usage with less standard tokens.

Recommendation	Consider using the current standard of handling deposits, which is based on how Uniswap handles transfer fees:
	
	`uint256 balanceBefore = pool.lpToken.balanceOf(address(this));`
	`pool.lpToken.transferFrom(msg.sender, address(this), _amount);`
	`_amount =`
	`pool.lpToken.balanceOf(address(this)).sub(balanceBefore);`
	
	Note that by using this method, you can also add the specific transfer tax logic for the native token.

Resolution	PARTIALLY RESOLVED The client has stated they will not add in tokens with transfer taxes.

Issue #06	Duplicated pools may be added to the Masterchef
Severity	MEDIUM SEVERITY
Description	The add function allows for duplicate pools to be added, which would lead to dilution of emission rewards to stakers.
Recommendation	The addition of a modifier that checks for duplicate pools could help prevent this incident from occuring.

```
mapping(IBEP20 => bool) public poolExistence;

modifier nonDuplicated(IBEP20 _lpToken) {
    require(poolExistence[_lpToken] == false, "nonDuplicated: duplicated");
    _;
}

// Add a new lp to the pool. Can only be called by the owner.
function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, bool
_withUpdate) external onlyOwner nonDuplicated(_lpToken) {

    require(_depositFeeBP <= 10000, "add: invalid deposit fee basis points");

    if (_withUpdate) {
            massUpdatePools();
    }

    uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    poolExistence[_lpToken] = true;

    poolInfo.push(PoolInfo({
            lpToken: _lpToken,
            allocPoint: _allocPoint,
            lastRewardBlock: lastRewardBlock,
            accH2OPerShare: 0,
            depositFeeBP: _depositFeeBP
    }));

    updateStakingPool();
}
```

Alternatively, you could account for this by adding in an **`lpSupply`** variable under **`poolInfo`**. This has the benefit of accurately accounting for deposits in the Masterchef.

Resolution	PARTIALLY RESOLVED The client has stated that they will not add in duplicate pools.

Issue #07	`emergencyWithdraw` function is vulnerable to reentrancy
Severity	MEDIUM SEVERITY
Description	Add this `nonReentrant` feature, and arranging the order such that user balances are set to 0 before transfers are made would be a safe practice in line with the Check Effects Interactions pattern.

Recommendation	To implement the Check Effects Interactions pattern :
	
	`function emergencyWithdraw(uint256 _pid) public {`
	`  PoolInfo storage pool = poolInfo[_pid];`
	`  UserInfo storage user = userInfo[_pid][msg.sender];`
	`  uint256 amount = user.amount;`
	`  user.amount = 0;`
	`  user.rewardDebt = 0;`
	`  pool.lpToken.safeTransfer(address(msg.sender), amount);`
	`  emit EmergencyWithdraw(msg.sender, _pid, amount);`
	`}`
	
	Alternatively, by adding Reentrancy Guard :
	
	`function emergencyWithdraw(uint256 _pid) external nonReentrant {`
	`  PoolInfo storage pool = poolInfo[_pid];`
	`  UserInfo storage user = userInfo[_pid][msg.sender];`
	`  pool.lpToken.safeTransfer(address(msg.sender), user.amount);`
	`  emit EmergencyWithdraw(msg.sender, _pid, user.amount);`
	`  user.amount = 0;`
	`  user.rewardDebt = 0;`
	`}`

Resolution	ACKNOWLEDGED

Issue #08	Minting will break if the multiplier is set to a very high value
Severity	LOW SEVERITY
Description	If the multiplier is set to an extremely large value using `updateMultiplier`, `SafeMath` will make the `updatePool` calls fail due to overflow. This will break any minting, depositing and withdrawing.
Recommendation	Consider adding a safeguard to the `updateMultiplier` function with a reasonable limit: `require(multiplierNumber < MAX_MULTIPLIER);`
Resolution	RESOLVED Resolved using suggested logic in the SafeOwner contract.

Issue #09	Adding a EOA or non-token contract as a pool will break `updatePool` and `massUpdatePools`
Severity	LOW SEVERITY
Description	`updatePool` will always call `balanceOf(address(this))` on the token of this pool.
Recommendation	Consider simply adding a test line in the `add` function. If the token does not exist, this will make sure the `add` function fails. `_lpToken.balanceOf(address(this));`
Resolution	RESOLVED Resolved using suggested logic in the SafeOwner contract.

Issue #10	dev function can be renamed, made `external`, and only be called by the owner of the contract
Severity	INFORMATIONAL
Description	Users may be confused on what the `dev` function does.
Recommendation	Consider renaming the function to `setDevAddress`, set the function to `external` and callable by `onlyOwner`.
Resolution	ACKNOWLEDGED

Issue #11	Pools use the contract balance to figure out the total deposits
Severity	INFORMATIONAL
Description	As with pretty much all Masterchefs, the total number of tokens in the Masterchef contract is used to determine the total number of deposits. This can cause dilution of rewards when people accidentally send tokens to the Masterchef. More severely, because the native token is constantly minted, this will cause severe dilution on the native token pool. This dilution is amplified even a bit more because the native token in question is a reflection token.
Recommendation	Consider adding an `lpSupply` variable to the PoolInfo that keeps track of the total deposits.
Resolution	ACKNOWLEDGED

Issue #12	`deposit`, and `withdraw`, `emergencyWithdraw`, `add`, and `set` can be made `external`
Severity	INFORMATIONAL
Description	The `deposit`, and `withdraw`, `emergencyWithdraw`, `add` and `set` 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 #13	`enterStaking` and `leaveStaking` are redundant
Severity	INFORMATIONAL
Description	Since the project has removed the syrup token, there is no code difference between `deposit` and `enterStaking`. There is also no code difference between `withdraw` and `leaveStaking`. It is good practice to remove redundant code to reduce the burden for future reviewers.
Recommendation	Consider removing `enterStaking` and `leaveStaking` to reduce code size.
Resolution	ACKNOWLEDGED

Issue #14	Lack of events for `add`, `set`, `setDevAddress` and `updateMultiplier`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	ACKNOWLEDGED

2.3 SmartChef

Stake syrup to earn reward tokens. Syrup could be `init` as any token and be used as an alternative, really low reward pool (see below about rewards).

2.3.1 Issues & Recommendations

Issue #15	Severely excessive rewards issue when a token with a transfer tax is added
Severity	MEDIUM SEVERITY
Description	When `_syrup` is set as a token with a transfer tax, this will result in significant excessive rewards due to the way the masterchef handles the reward mechanism. Please see the relevant masterchef issue for more information. This issue is only marked as medium severity as we are assuming the native token will be used for `_syrup`, which does not have a transfer tax.

Recommendation	Consider using the current standard of handling deposits, which is based on how Uniswap handles transfer fees:
	
	`uint256 balanceBefore = pool.lpToken.balanceOf(address(this));`
	`pool.lpToken.transferFrom(msg.sender, address(this), _amount);`
	`_amount =`
	`pool.lpToken.balanceOf(address(this)).sub(balanceBefore);`
	
	Note that by using this method, you can also add the specific transfer tax logic for the native token.

Resolution	ACKNOWLEDGED

Issue #16	Rewards can be stopped by owner at any time
Severity	MEDIUM SEVERITY
Description	Calling the `stopReward` function will set multipliers to 0.
Recommendation	Remove this function if it's not needed. Otherwise, if there is a justifiable reason, then consider transferring ownership of the SmartChef contract to a minimum 2 day Timelock.
Resolution	ACKNOWLEDGED

Issue #17	Withdrawing not possible in certain scenarios
Severity	MEDIUM SEVERITY
Description	`withdraw` will revert if there are not enough reward tokens in the SmartChef contract. This would occur if `emergencyRewardWithdraw` was called, or too little reward tokens are sent to the contract.
Recommendation	Consider using `safeRewardTransfer.`
Resolution	ACKNOWLEDGED

Issue #18	`emergencyWithdraw` function is vulnerable to reentrancy
Severity	MEDIUM SEVERITY
Description	Add this **`nonReentrant`** feature, and arranging the order such that user balances are set to 0 before transfers are made would be a safe practice in line with the Check Effects Interactions pattern (https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html).

Recommendation	To implement the Check Effects Interactions pattern :
	
	`function emergencyWithdraw(uint256 _pid) public {`
	`  PoolInfo storage pool = poolInfo[_pid];`
	`  UserInfo storage user = userInfo[_pid][msg.sender];`
	`  uint256 amount = user.amount;`
	`  user.amount = 0;`
	`  user.rewardDebt = 0;`
	`  pool.lpToken.safeTransfer(address(msg.sender), amount);`
	`  emit EmergencyWithdraw(msg.sender, _pid, amount);`
	`}`
	
	Alternatively, by adding Reentrancy Guard :
	
	`function emergencyWithdraw(uint256 _pid) external nonReentrant {`
	`  PoolInfo storage pool = poolInfo[_pid];`
	`  UserInfo storage user = userInfo[_pid][msg.sender];`
	`  pool.lpToken.safeTransfer(address(msg.sender), user.amount);`
	`  emit EmergencyWithdraw(msg.sender, _pid, user.amount);`
	`  user.amount = 0;`
	`  user.rewardDebt = 0;`
	`}`

Resolution	ACKNOWLEDGED

Issue #19	Lack of events for `emergencyRewardWithdraw` and `stopRewards`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Consider adding an event for these functions.
Resolution	ACKNOWLEDGED

Issue #20	`emergencyRewardWithdraw`, `emergencyWithdraw`, `withdraw` and `deposit` can be made `external`.
Severity	INFORMATIONAL
Description	The `emergencyRewardWithdraw`, `emergencyWithdraw`, `withdraw` and `deposit` 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(s)	Consider making these functions `external`.
Resolution	ACKNOWLEDGED

Issue #21	Constructor does not have parameter safeguards
Severity	INFORMATIONAL
Description	Checks for `startBlock` and `bonusEndBlock` to be in the future, and for the syrup token to not be the same as the reward token. Setting the reward token to be the same as syrup token could lead to undesirable results.

Recommendation	Consider adding the following checks to the constructor :
	`require(_startBlock > block.number);`
	`require(_bonusEndBlock > _startBlock);`
	`require(_syrup != _rewardToken);`

Resolution	ACKNOWLEDGED

2.4 IFO

The IFO contract represents a fund raising pool where users can stake `lpTokens` in the deposit round with the intention of receiving `offeringTokens` in the harvest round. While this contract is based on PancakeSwap's obsolete IFO.sol, it extends it by actually allocating the funds in an underlying masterchef, similar to a vault.

When the deposit round is finished, no more deposits can be made. If the maximum raising amount was not reached, everyone receives offering tokens according to the advertised price. If the maximum raising amount is exceeded, a part of the raising tokens is refunded pro-rata to maintain the advertised IFO price.

2.4.1 Privileged Roles

The following `onlyOwner` functions can be called:

- `setOfferingAmount`
- `setRaisingAmount`
- `updateStartAndEndBlocks`
- `finalWithdrawLPToken`
- `finalWithdrawOfferingToken`

2.4.2 Issues & Recommendations

Issue #22	Admin can withdraw `lpTokens` meant for refunding after IFO ends, causing all funds to be stuck in the contract
Severity	HIGH SEVERITY
Description	If the IFO raises a high number of funds, the users will receive a partial refund of their funds after the IFO is finished. However, since the admin can withdraw all LP tokens this can block harvests, as no balance check is made there.
Recommendation(s)	Consider adding a significant delay (eg. 7 days) to the `finalWithdrawLPToken` or updating the business logic so that this event can never occur.
Resolution	RESOLVED The client has added a 7 day delay to the function.

Issue #23	User allocation is only precise up to 1/1000000
Severity	HIGH SEVERITY
Description	The user allocation represents the percentage of the pool owned by the user. This is used for calculating rewards and how much `offeringTokens` should be given to the user. However, since this number only has a precision of 1/10^6, it might not be precise, causing the users to receive too few tokens. For example, if there are 10 million `lpTokens` staked, anything less than $10 would be seen as a stake of $0 and all stakes are rounded down to the nearest multiple of 10. This issue presents itself in the calculation of the refunding amount as well, but in another manner: here, the amount is actually rounded up. In the above example with 10 million `lpTokens`, if a user were to stake $9.99 their allocation would still be zero and they would be refunded the complete $9.99. Note that in case no refund would be in order, these users would not receive any tokens at all.
Recommendation(s)	Consider increasing the precision of the `userAllocation` variable.
Resolution	ACKNOWLEDGED

Issue #24	No sanity checks in the constructor can lead to serious problems in the business logic
Severity	MEDIUM SEVERITY
Description	The constructor does not have any sanity checks which can lead to severe miscalculations in the business logic. For example, an IFO could be created with `lpToken` being the same as either the `offeringToken` or `farmRewardToken`. This would, for example, later cause issues where the `farmRewardToken` would be distributed to users, allowing the early harvesters to take out the `lpTokens` that should go to the project owner. Another example is that if certain integer values are set to zero, it will cause severe malfunction of the contract because SafeMath will revert any division by these values.
Recommendation(s)	Consider adding reasonable sanity checks (upper bounds, lower bounds and inequalities) to all constructor paramaters so future reviewers can simply check the code and don't have to carefully check each parameter.
Resolution	RESOLVED

Issue #25	Deposit amounts will be miscalculated for tokens with a transfer tax before `approve` is called and will stop working once `approve` is called
Severity	MEDIUM SEVERITY
Description	If the IFO has a token with a transfer tax as the lpToken, it will cause the pool to fail if it deposits in the underlying masterchef because it cannot deposit sufficient funds in the underlying masterchef. If the approve method has not yet been called it will cause the balances to be incorrect with the actual received tokens.

Recommendation(s)	Consider using the current standard of handling deposits, which is based on how Uniswap handles transfer fees:
	`uint256 balanceBefore = pool.lpToken.balanceOf(address(this));`
	`pool.lpToken.transferFrom(msg.sender, address(this), _amount);`
	`_amount = pool.lpToken.balanceOf(address(this)).sub(balanceBefore);`
	Note that by using this method, you can also add the specific transfer tax logic for the native token.

Resolution	RESOLVED Resolved using suggested logic.

Issue #26	The IFO contract might stake in an insecure masterchef
Severity	LOW SEVERITY
Description	During the deployment of the IFO contract, the underlying masterchef can be chosen. In case the underlying masterchef is an untrusted masterchef, this could lead to loss of funds.
Recommendation(s)	Consider clearly indicating to the users in which underlying protocol their funds are being staked.
Resolution	ACKNOWLEDGED

Issue #27	Harvesting could be blocked in a number of cases
Severity	LOW SEVERITY
Description	The harvest method could revert in certain cases, making the harvests/withdrawal step impossible. Here is a list of cases where the harvest will fail: 1. In case the farm cannot be ended (for example when a token with an anti-whale feature has been staked in the underlying masterchef and cannot be withdrawn) 2. in case there are insufficient `lpTokens` in the masterchef (eg. withdrawn by admin, as discussed above) 3. in case there are insufficient `farmRewardTokens` in the masterchef (could be caused by the `rewardToken` having a transfer tax) 4. in case the offering token reverts on a zero amount transfer and the allocation rounding causes the user to receive zero offering tokens.
Recommendation(s)	Consider making sure that farming can always end and adding a `transfer` function which will never fail, similar to the `safeRewardTransfer` functions in a Masterchef. These functions simply transfer the balance of the contract if there are insufficient tokens. Finally a greater than zero check should be considered on the `offeringToken` transfer.
Resolution	PARTIALLY RESOLVED The client implemented most of our recommendations but did not add a greater than zero check on the `offeringToken` transfer.

Issue #28	`lpToken`, `offeringToken`, `masterchef`, `pid` and `farmRewardToken` can be made immutable
Severity	INFORMATIONAL
Description	Variables that are set in the constructor but never changed throughout the lifecycle of the contract can be made immutable to indicate this to third-party reviewers. This will make it easier for the reviewers to understand the code.
Recommendation(s)	Consider making these variables immutable.
Resolution	ACKNOWLEDGED

Issue #29	Lack of events for `setOfferingAmount`, `setRaisingAmount`, `updateStartAndEndBlocks`, `approve`, `_endFarmStaking`, `finalWithdrawLPToken` and `finalWithdrawOfferingToken`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation(s)	Add events for these functions.
Resolution	ACKNOWLEDGED

2.5 Timelock

The timelock is a standard clone of Compound Finance's timelock.

Parameter	Value	Description
Delay	1 day	The `delay` indicates the time the administrator has to wait after queuing a transaction to execute it.
Minimum Delay	6 hours	The `minDelay` indicates the lowest value that the `delay` can minimally be set. Sometimes, projects will queue a transaction that sets the `delay` to zero with the hope that nobody notices it. However, because of the minimum delay parameter, the value of `delay` can never be lower than that of the `minDelay` value. Note that the administrator could still queue a transaction to simply transfer the ownership back to their own account so it is still important to inspect every transaction carefully.
Grace Period	14 days	After the delay has expired after queueing a transaction, the administrator can only execute it within the grace period. This is to prevent them from hiding a malicious transaction among much earlier transactions, hoping that it goes unnoticed or buried, which can be executed in the future.

2.5.1 Issues & Recommendations

No issues found.


PALADIN
BLOCKCHAIN SECURITY