PALADIN
BLOCKCHAIN SECURITY

# Smart Contract Security Assessment

Final Report

## For FlatSwap Exchange

09 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 FlatSwap Exchange 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** | FlatSwap Exchange |
| **URL** | https://flatswap.exchange/ |
| **Platform** | Binance Smart Chain |
| **Language** | Solidity |

## 1.2 Contracts Assessed

| Name | Contract | Live Code Match |
|---|---|---|
| `CoinToken` | `0xd22246644d2BE5d0427a8E474477d96677C3eC24` | MATCH |
| `MasterChef` | `0x7Df83cd8CA76D89A69f3aB30aA11295a3F502488` | MATCH |
| `FSwapReferral` | `0x6622D0c880c2670f860C312f05Df35b7eA495923` | MATCH |
| `FSwapPoolInitializable` | `0x8c984D4eF008acf291d9F68CD814e0C65F080dEb` | MATCH |
| `Timelock` | `0xadb098c74dfb888b4b4457e6e7d5e5b431d12d37` | MATCH |

## 1.3 Findings Summary

| Severity | Found | Resolved | Partially Resolved | Acknowledged (no change made) |
|---|---|---|---|---|
| High | 1 | 1 | - | - |
| Medium | 6 | 2 | 1 | 3 |
| Low | 4 | 1 | - | 3 |
| Informational | 21 | 8 | 1 | 12 |
| **Total** | **32** | **12** | **2** | **18** |

### 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 CoinToken

| 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 | `feeReceiver` logic in constructor is unnecessary | ACKNOWLEDGED |

### 1.3.2 MasterChef

| ID | Severity | Summary | Status |
|---|---|---|---|
| 03 | HIGH | Masterchef calls function on native token that doesn't exist on deployed native token contract | RESOLVED |
| 04 | MEDIUM | Transfer-tax mitigation could be misused or accidentally set to incorrect values | ACKNOWLEDGED |
| 05 | LOW | `updateEmissionRate` has no maximum safeguard | ACKNOWLEDGED |
| 06 | INFORMATIONAL | Pools use the contract balance to figure out the total deposits | ACKNOWLEDGED |
| 07 | INFORMATIONAL | Adding an EOA or a non-token contract as a pool will break `updatePool`, `massUpdatePools` and `updateEmissionRate` | ACKNOWLEDGED |
| 08 | INFORMATIONAL | The pendingFSwap function will revert if totalAllocPoint is zero | ACKNOWLEDGED |
| 09 | INFORMATIONAL | There are no sanity checks in the setReferralAddress function | ACKNOWLEDGED |
| 10 | INFORMATIONAL | Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions | ACKNOWLEDGED |
| 11 | INFORMATIONAL | `add`, `set`, `deposit`, `withdraw`, `emergencyWithdraw`, `updateEmissionRate`, `setDevAddress`, `setFeeAddress` and `setTokenTaxRate` can be made `external` | ACKNOWLEDGED |
| 12 | INFORMATIONAL | Lack of events for `setTokenTaxRate`, `setFSwapReferral`, `setReferralCommissionRate`, `setDevAddress`, `setFeeAddress`, `add` and `set` | ACKNOWLEDGED |
| 13 | INFORMATIONAL | Comment for deposit fee cap does not match code | ACKNOWLEDGED |
| 14 | INFORMATIONAL | `fswap` and `startBlock` can be made `immutable` | ACKNOWLEDGED |

### 1.3.3 FSwapReferral

| ID | Severity | Summary | Status |
|---|---|---|---|
| 15 | LOW | The owner of the FSwapReferral contract can overwrite themselves as the referrer for all users using the `recordReferral` function | ACKNOWLEDGED |
| 16 | INFORMATIONAL | `recordReferral` function can be made `external` | ACKNOWLEDGED |

### 1.3.4 FSwapPoolInitializable

| ID | Severity | Summary | Status |
|---|---|---|---|
| 17 | MEDIUM | Transfer-tax mitigation could be misused or accidentally set to incorrect values | RESOLVED |
| 18 | MEDIUM | The owner can take all reward tokens out (not staked tokens) | PARTIAL |
| 19 | MEDIUM | Rewards can be stopped by owner at any time | ACKNOWLEDGED |
| 20 | MEDIUM | Withdrawing not possible in certain scenarios | RESOLVED |
| 21 | LOW | Setting `feeAddress` to the zero address will prevent depositing | RESOLVED |
| 22 | INFORMATIONAL | There is no validation in the constructor to check whether `startBlock` must be lower than `bonusEndBlock` and whether `_stakedToken` and `_rewardToken` are valid token addresses, and are not the same address | RESOLVED |
| 23 | INFORMATIONAL | Calling `stopReward` after the pool has stopped will continue to extend the reward period | RESOLVED |
| 24 | INFORMATIONAL | `EmergencyWithdraw` event will always be emitted with a zero amount | RESOLVED |
| 25 | INFORMATIONAL | Unnecessary `realAmount` adjustment in `withdraw` | RESOLVED |
| 26 | INFORMATIONAL | Lack of events for `emergencyRewardWithdraw` and `stopRewards` | RESOLVED |
| 27 | INFORMATIONAL | `stakedToken`, `rewardToken` and `PRECISION_FACTOR` can be made immutable | ACKNOWLEDGED |
| 28 | INFORMATIONAL | `msg.sender` is unnecessarily cast to an address | RESOLVED |
| 29 | INFORMATIONAL | `updateRewardPerBlock` has no maximum safeguard | PARTIAL |
| 30 | INFORMATIONAL | Pool uses the contract balance to figure out the total deposits | RESOLVED |
| 31 | INFORMATIONAL | Maximum deposit and transfer-tax checks use < instead of <= | RESOLVED |

### 1.3.5 Timelock

| ID | Severity | Summary | Status |
|---|---|---|---|
| 32 | MEDIUM | Admin can update `delay` at any time | ACKNOWLEDGED |

# 2 Findings

## 2.1 CoinToken

The CoinToken contract is an ERC20-compatible token based on standard openzeppelin contracts, with additional functionality for ERC165, ERC1363, burning and token recovery. It bears a strong resemblance to tokens generated using the vittominacori/token-generator project and the ERC1363 implementation is identical to the one at vittominacori/erc1363-payable-token.

The only modification of standard libraries is that `_owner` has been made public on `Ownable` and is set directly in the `CoinToken` constructor as a separate address to `msg.sender`.

Besides the usual approve/transfer functionality of a standard ERC20 token, the ERC1363 extensions add new functions: `approveAndCall`, `transferAndCall` and `transferFromAndCall`. These functions allow a callback contract address to be passed in that must implement `onApprovalReceived` or `onTransferReceived`, which will be called after the approve/transfer has completed. This opens up a number of use cases, though care needs to be taken when using them to ensure there are no re-entrancy issues. None of the contracts we reviewed call these functions.

Finally, the contract allows the owner to take out tokens sent to it by accident using the `recoverERC20` function. This function does not allow the owner to take tokens out of other contracts and is therefore innocent.

### 2.1.1 Token Overview

| | |
|---|---|
| **Address** | 0xd22246644d2BE5d0427a8E474477d96677C3eC24 |
| **Name** | FlatSwap |
| **Symbol** | FSWAP |
| **Initial Supply** | 20,000,000 minted to 0x0d66d73fe5c375cbc673c72a97c91cea66800f4f |
| **Token Supply** | Unlimited |
| **Decimal Places** | 18 |
| **Transfer Max Size** | None |
| **Transfer Min Size** | None |
| **Transfer Fees** | None |

### 2.1.2 Privileged Roles

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

- `renounceOwnership`
- `transferOwnership`
- `recoverERC20`
- `mint`
- `finishMinting`

The ownership of the contract has already been transferred to the Masterchef, and the only function of these it is coded to call is `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** | 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.<br><br>As token ownership has already been transferred to the Masterchef, the main risk that remains is whether any tokens were pre-minted to the project owner prior to transferring ownership. |
| **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** | ACKNOWLEDGED |

| Issue #02 | `feeReceiver` logic in constructor is unnecessary |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | The token constructor transfers any MATIC sent with the contract creation transaction to a `feeReceiver` address.<br><br>It is possible this was used to pay a third-party provider for the service of setting up the token, or that the code was copied from a token that had been setup this way. However, it's unnecessary for the functionality of the token. |
| **Recommendation** | Consider removing this functionality. |
| **Resolution** | ACKNOWLEDGED |

## 2.2 MasterChef

The MasterChef is a straightforward Panther fork with allowances for transfer-tax pools beyond just a native token pool. Deposit fees are limited to at most 21.6%[1] while the referral commission can be set up to 10%. Harvest intervals are capped at 14 days.

### 2.2.1 Privileged Roles

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

- `renounceOwnership`
- `transferOwnership`
- `add`
- `set`
- `setTokenTaxRate`
- `updateEmissionRate`
- `setFSwapReferral`
- `setReferralCommissionRate`

[1] There is a 2% limit on the amount that can be sent to the `feeAddress`, but each pool has a transfer-tax subtraction setting of up to 20%, so 21.6% total could be subtracted from each user's deposit.

## 2.2.2 Issues & Recommendations

| Issue #03 | **Masterchef calls function on native token that doesn't exist on deployed native token contract** |
|---|---|
| **Severity** | HIGH SEVERITY |
| **Description** | The FSwapToken contract code included in the Masterchef contract does not match the deployed CoinToken contract, the address of which was passed into the Masterchef constructor. Specifically, the deployed CoinToken contract has no `transferTaxRate()` function, which is called from the Masterchef's `deposit()` function. This will result in deposits failing on the native FSWAP pool (`pid` 8). |
| **Recommendation** | Consider removing any calls to `fswap.transferTaxRate()` and replacing them with alternative means of calculating any native token transfer tax as indicated in the following issue (note that there is no native token transfer tax). Also consider including the actual FSWAP token contract code in the Masterchef as this would allow the Solidity compiler to catch any implementation differences.<br><br>If redeploying a new Masterchef is not an option, it is highly recommended to set the `allocPoint` of the FSWAP pool (`pid` 8) to 0 and remove the pool from the website's interface. |
| **Resolution** | RESOLVED<br><br>The client has implemented the recommendation to set the `allocPoint` of the FSWAP pool (pid 8) to 0. |

| Issue #04 | Transfer-tax mitigation could be misused or accidentally set to incorrect values |
|---|---|
| **Severity** | MEDIUM SEVERITY |
| **Description** | In many forks of SushiSwap, mismatches in balances due to transfer-tax token pools can be exploited, resulting in significant excessive rewards being minted.<br><br>FlatSwap has taken steps to address this for which we commend them. The Masterchef allows for per-token values in a `_taxableTokens` mapping that can be used to adjust deposited balances to account for transfer tax tokens pools.<br><br>However, this implementation requires precise manual setting for each token by the Masterchef owner, and needs to be kept in sync with any transfer tax changes in the tokens themselves. This is especially difficult if the Masterchef owner is a timelock.<br><br>It could also be misused to subtract funds from users' deposits if it is set higher than the actual transfer-tax on the token. |
| **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);
```

| | |
|---|---|
| **Resolution** | ACKNOWLEDGED |

| Issue #05 | `updateEmissionRate` has no maximum safeguard |
|---|---|
| **Severity** | LOW SEVERITY |
| **Description** | Projects sometimes accidentally update their emission rate to a severely high number either by accident or with malicious intent. |
| **Recommendation** | Consider adding a `MAX_EMISSION_RATE` variable and setting it to a reasonable value.<br>`require(_fswapPerBlock <= MAX_EMISSION_RATE,"Too high");` |
| **Resolution** | ACKNOWLEDGED |

| Issue #06 | 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. |
| **Recommendation** | Consider adding an `lpSupply` variable to the `PoolInfo` that keeps track of the total deposits. |
| **Resolution** | ACKNOWLEDGED |

| Issue #07 | Adding an EOA or a non-token contract as a pool will break `updatePool`, `massUpdatePools` and `updateEmissionRate` |
|---|---|
| **Severity** | LOW SEVERITY |
| **Description** | `updatePool` will always call `balanceOf(address(this))` on the token of this pool, and will fail if the token is not an actual token contract address. |
| **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.<br><br>`_lpToken.balanceOf(address(this));` |
| **Resolution** | ACKNOWLEDGED |

| Issue #08 | The `pendingFSwap` function will revert if `totalAllocPoint` is zero |
|---|---|
| **Severity** | LOW SEVERITY |
| **Description** | In the `pendingFSwap` function, at some point a division is made by the `totalAllocPoint` variable. If all pools have their rewards set to zero, this variable will be zero as well. The requests will then revert with a division by zero error. |
| **Recommendation** | Consider only calculating the accumulated rewards since the `lastRewardBlock` if the `totalAllocPoint` variable is greater than zero.<br><br>This check can simply be added to the existing check that verifies the `block.number` and `lpSupply`, like so:<br><br>`if (block.number > pool.lastRewardBlock && lpSupply != 0 && totalAllocPoint > 0) {` |
| **Resolution** | ACKNOWLEDGED |

| Issue #09 | There are no sanity checks in the `setReferralAddress` 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. |
| **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 #10 | Rounding vulnerability to tokens with a very large supply can cause large supply tokens to receive zero emissions |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Within `updatePool`, `accFSwapPerShare` is based on the `lpSupply` variable.<br>`pool.accFSwapPerShare = pool.accFSwapPerShare.add(fswapReward.mul(1e12).div(lpSupply));`<br>However, if this `lpSupply` becomes a severely large value, precision errors may occur due to rounding. This is famously seen when pools 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 #11 | **`add`, `set`, `deposit`, `withdraw`, `emergencyWithdraw`, `updateEmissionRate`, `setDevAddress`, `setFeeAddress` and `setTokenTaxRate` 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 #12 | **Lack of events for `setTokenTaxRate`, `setFSwapReferral`, `setReferralCommissionRate`, `setDevAddress`, `setFeeAddress`, `add` and `set`** |
|---|---|
| **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 |

| Issue #13 | Comment for deposit fee cap does not match code |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | The following lines contradict each other:<br>`// Max deposit fee: 10%`<br>`uint16 public constant MAXIMUM_DEPOSIT_FEE = 200;` |
| **Recommendation** | Consider updating the comment to match the value in the code (2%). |
| **Resolution** | ACKNOWLEDGED |

| Issue #14 | `fswap` and `startBlock` 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. |
| **Recommendation** | Consider making the above variables explicitly `immutable`. |
| **Resolution** | ACKNOWLEDGED |

## 2.3 FSwapReferral

The FSwapReferral contract is a clean fork of the Pantherswap Referral contract. The deployed Masterchef currently has the zero address as its referral, so it is not utilizing this functionality and no referrals are currently being recorded or paid.

Finally, the contract allows the owner to take out tokens sent to it by accident using the `drainBEP20Token` function. This function does not allow the owner to take tokens out of other contracts and is therefore innocent.

### 2.3.1 Privileged Roles

The Masterchef should be an operator of the FSwapReferral contract. The following `onlyOperator` functions can be called by the Masterchef:

- `recordReferral`
- `recordReferralCommission`

The owner of the FSwapReferral contract is an EOA. The following `onlyOwner` functions can be called by the EOA::

- `updateOperator`
- `drainBEP20Token`

### 2.3.2 Issues & Recommendations

| Issue #15 | **The owner of the FSwapReferral contract can overwrite themselves as the referrer for all users using the `recordReferral` function** |
|---|---|
| **Severity** | LOW SEVERITY |
| **Description** | The operator should only be the Masterchef contract. However, the owner of the FSwapReferral contract has privileges that may be abused.<br><br>The following steps detail how the owner can make themselves the referrer for all users :<br><br>1. Owner of the FSwapReferral contract calls `updateOperator` to add themselves as an operator.<br>2. As an operator, they then call `recordReferral` with their own wallet address as the referrer for each user.<br>3. This results in their wallet address being minted potentially large amounts of referral commission. |
| **Recommendation** | There are 3 possible recommendations :<br><br>1. Consider making the `updateOperator` function callable only once.<br>2. Setting only the Masterchef as an operator in the modifier, and removing the `updateOperator` function.<br>3. Calling the `updateOperator` to set the Masterchef as operator, then renouncing ownership of the FSwapReferral contract such that it cannot be called in the future. |
| **Resolution** | ACKNOWLEDGED |

| Issue #16 | `recordReferral` function can be made external |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | The `recordReferral` 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** | ACKNOWLEDGED |

## 2.4 FSwapPoolInitializable

FSwapPoolInitializable is a staking pool contract forked from PancakeSwap's SmartChefInitializable with the addition of deposit fees and support for tokens with transfer taxes. Initially the FSWAP-DOGE pool was audited, which has deposit fees limited to at most 36%[2]. During the audit process, a new FSWAP-CAKE pool was deployed, which resolved many issues and has deposit fees limited to 20%.

As with PancakeSwap's staking pools, users can stake one type of token (`stakedToken`) and earn rewards in another token (`rewardToken`) at a particular rate (`rewardPerBlock`). A limit can be imposed on the pool to prevent users from staking more than a particular amount in total, and this limit can be increased or turned off by the owner (it cannot be decreased).

Finally, the contract allows the owner to take out tokens sent to it by accident using the `recoverWrongTokens` function. This function does not allow the owner to take out the staked token or the reward token and is therefore innocent.

[2] There is a 20% limit on the amount that can be sent to the `feeAddress`, but each pool has a transfer-tax subtraction setting of up to 20%, so 36% total could be subtracted from each user's deposit.

### 2.4.1 Privileged Roles

The owner of the FSwapPoolInitializable contract is an EOA/Timelock (TBD). The following `onlyOwner` functions can be called by the EOA/Timelock (TBD):

- `renounceOwnership`
- `transferOwnership`
- `emergencyRewardWithdraw`
- `recoverWrongTokens`
- `stopReward`
- `updatePoolLimitPerUser`
- `updateRewardPerBlock`
- `updateTokenTaxRate`
- `updateDepositFee`
- `updateFeeAddress`
- `updateStartAndEndBlocks`

### 2.4.2 Issues & Recommendations

| Issue #17 | **Transfer-tax mitigation could be misused or accidentally set to incorrect values** |
|---|---|
| **Severity** | MEDIUM SEVERITY |
| **Description** | As with the Masterchef, FlatSwap have taken steps to account for staking transfer-tax tokens. The staking pool allows for a rate to be set to adjust deposited balances to account for a transfer-tax on the staking token.<br><br>However, this implementation requires precise manual setting by the staking pool owner, and needs to be kept in sync with any transfer tax changes in the tokens themselves. This is especially difficult if the staking pool owner is a timelock.<br><br>It could also be misused to subtract funds from users' deposits if it is set higher than the actual transfer-tax on the token. |
| **Recommendation** | Consider using the current standard of handling deposits, which is based on how Uniswap handles transfer fees:<br><br>`uint256 balanceBefore = stakedToken.balanceOf(address(this));`<br>`stakedToken.safeTransferFrom(address(msg.sender), address(this), _amount);`<br>`_amount = stakedToken.balanceOf(address(this)).sub(balanceBefore);`<br>`user.amount = user.amount.add(_amount);`<br><br>This method accounts for any transfer-tax and does not require maintaining extra state values that need to be set or updated. |
| **Resolution** | RESOLVED<br><br>The client has implemented the recommendation. |

| Issue #18 | The owner can take all reward tokens out (not staked tokens) |
|---|---|
| **Severity** | MEDIUM SEVERITY |
| **Description** | The owner of the contract can call `emergencyRewardWithdraw` to remove all reward tokens.<br><br>If a pool has deposit fees that users have paid when staking, there's a reasonable expectation that they'll make back at least some of the value of this deposit fee with rewards. If the owner removes all the rewards, then this will no longer be possible. |
| **Recommendation** | Remove this function if it's not needed. Otherwise, if there is a justifiable reason, then consider only allowing rewards to be withdrawn after the reward period has ended, and only allow the reward period to be set once (or behind a timelock as suggested in the following issue) |
| **Resolution** | PARTIALLY RESOLVED<br><br>The client has implemented the recommendation that `emergencyRewardWithdraw` cannot be called while the reward period is active. They did not implement the recommendation that the reward period can only be set once. So users should note that the owner of the contract can still stop rewards at any time and then call this function. |

| Issue #19 | 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 FSwapPoolInitializable contract to a minimum 2-day Timelock. |
| **Resolution** | ACKNOWLEDGED |

| Issue #20 | Withdrawing not possible in certain scenarios |
|---|---|
| **Severity** | MEDIUM SEVERITY |
| **Description** | `withdraw` will revert if there are not enough reward tokens in the FSwapPoolInitializable contract. This would occur if `emergencyRewardWithdraw` was called, or too little reward tokens are sent to the contract. |
| **Recommendation** | Consider using `safeRewardTransfer`, similar to how it is used in the Masterchef. |
| **Resolution** | RESOLVED<br>The client has implemented the recommendation. |

| Issue #21 | Setting `feeAddress` to the zero address will prevent depositing |
|---|---|
| **Severity** | LOW SEVERITY |
| **Description** | Transferring tokens to the zero address will revert transactions. Deposits will thus revert if the `feeAddress` is ever set to the zero address.<br><br>As this would only prevent deposits and not withdrawals, it is not considered a high severity issue. |
| **Recommendation** | To prevent this from ever happening by accident and to limit governance risks, consider adding a requirement like<br><br>`require(_feeAddress != address(0), "!nonzero");`<br><br>to the `updateFeeAddress` function. |
| **Resolution** | RESOLVED<br>The client has implemented the recommendation. |

| Issue #22 | **There is no validation in the constructor to check whether `startBlock` must be lower than `bonusEndBlock` and whether `_stakedToken` and `_rewardToken` are valid token addresses, and are not the same address** |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | `_getMultiplier` will always return 0 if the `bonusEndBlock` is less than the `startBlock`.<br><br>`_updatePool` will always call `balanceOf(address(this))` on the `stakedToken`, and will fail if the token is not an actual token contract address.<br><br>Setting the reward token to be the same as the staked token could lead to undesirable results. |
| **Recommendation** | Consider adding these checks to the contract constructor:<br><br>`_stakedToken.balanceOf(address(this));`<br>`_rewardToken.balanceOf(address(this));`<br>`require(_stakedToken != _rewardToken, "stakedToken must be different to rewardToken");`<br>`require(_startBlock > block.number, "startBlock cannot be in the past");`<br>`require(_startBlock < _bonusEndBlock, "startBlock must be lower than endBlock");` |
| **Resolution** | RESOLVED<br><br>The client has implemented the recommendation. |

| Issue #23 | **Calling `stopReward` after the pool has stopped will continue to extend the reward period** |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | As `stopReward` updates the `bonusEndBlock`, calling this after the pool has already stopped will allow users to withdraw extra rewards which will cause a loss of the pool's assets. |
| **Recommendation** | Consider adding checks to ensure the pool has not already ended:<br>`function stopReward() external onlyOwner {`<br>`  require(startBlock < block.number, "Pool has not started");`<br>`  require(block.number <= bonusEndBlock, "Pool has ended");`<br>`  bonusEndBlock = block.number;`<br>`}` |
| **Resolution** | RESOLVED<br>The client has implemented the recommendation. |

| Issue #24 | **`EmergencyWithdraw` event will always be emitted with a zero amount** |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | `emergencyWithdraw` sets user.amount to 0, and then uses this value later when it emits the `EmergencyWithdraw` event.<br>`uint256 amountToTransfer = user.amount;`<br>`user.amount = 0;`<br>`user.rewardDebt = 0;`<br>`if (amountToTransfer > 0) {`<br>`  stakedToken.safeTransfer(address(msg.sender), amountToTransfer);`<br>`  }`<br>`emit EmergencyWithdraw(msg.sender, user.amount);` |
| **Recommendation** | Emit with `amountToTransfer` instead:<br>`emit EmergencyWithdraw(msg.sender, amountToTransfer);` |
| **Resolution** | RESOLVED<br>The client has implemented the recommendation. |

| Issue #25 | Unnecessary `realAmount` adjustment in `withdraw` |
| --- | --- |
| **Severity** | INFORMATIONAL |
| **Description** | In `withdraw`, there's a calculation to modify the actual amount withdrawn if the user tries to withdraw more than they have:<br><br>`if (_amount > 0) { uint256 realAmount = _amount; if (user.amount < realAmount) { realAmount = user.amount; } user.amount = user.amount.sub(realAmount); stakedToken.safeTransfer(address(msg.sender), realAmount); }`<br><br>However, this will never be the case because of a requirement earlier in withdraw:<br><br>`require(user.amount >= _amount, "Amount to withdraw too high");` |
| **Recommendation** | Consider removing the logic for calculating `realAmount` and just use `_amount`. If the user tries to withdraw more than they have, the transaction will revert, but this is to be expected.<br><br>If it is desired to not revert in such cases, then remove the earlier `require`, but also be sure to modify the `Withdraw` event emittal with the correct value transferred, as it currently just uses `_amount`. |
| **Resolution** | RESOLVED<br><br>The client has implemented the recommendation. |

| Issue #26 | **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** | RESOLVED<br>The client has implemented the recommendation. |

| Issue #27 | **`stakedToken`, `rewardToken` and `PRECISION_FACTOR` 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. |
| **Recommendation** | Consider making the above variables explicitly `immutable`. |
| **Resolution** | ACKNOWLEDGED |

| Issue #28 | `msg.sender` is unnecessarily cast to an address |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | In Solidity, the `msg.sender` literal is an address and thus does not need to be cast to an address manually. |
| **Recommendation** | Consider replacing `address(msg.sender)` with `msg.sender`. |
| **Resolution** | RESOLVED<br>The client has implemented the recommendation. |

| Issue #29 | `updateRewardPerBlock` has no maximum safeguard |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | Projects sometimes accidentally update their emission rate to a severely high number either by accident or with malicious intent.<br><br>This is less of an issue in this contract as it can only be adjusted before the rewards have started, hence it is Informational only. |
| **Recommendation** | Consider adding a `MAX_EMISSION_RATE` variable and setting it to a reasonable value.<br>`require(_rewardPerBlock <= MAX_EMISSION_RATE);` |
| **Resolution** | PARTIALLY RESOLVED<br>The client has implemented the recommendation, though users should note the maximum safeguard implemented is 10,000,000 tokens per block, which is still very high for most tokens. We do acknowledge that it's difficult to come up with a maximum safeguard that can cover all tokens. |

| Issue #30 | Pool uses the contract balance to figure out the total deposits |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | The total number of staked tokens in the contract is used to determine the total number of deposits. This can cause dilution of rewards when people accidentally send tokens to the contract.<br><br>This is unlikely to lead to as many issues as it would in the Masterchef that has multiple pools and minting, so it is provided purely for Informational purposes. |
| **Recommendation** | Consider adding an `lpSupply` variable that keeps track of the total deposits. |
| **Resolution** | RESOLVED<br><br>The client has implemented the recommendation. |

| Issue #31 | Maximum deposit and transfer-tax checks use < instead of <= |
|---|---|
| **Severity** | INFORMATIONAL |
| **Description** | In the constructor, `updateTokenTaxRate` and `updateDepositFee`, there are checks to ensure the given rates don't exceed the maximum:<br><br>`require(_tokenTaxRate < MAX_TOKEN_TAX_RATE, "Invalid token tax rate");`<br>`require(_depositFee < MAX_DEPOSIT_FEE, "Invalid deposit fee");`<br><br>Typically, and with maximum checks in the other contracts such as the Masterchef and Timelock, maximum caps are inclusive. That is, the value can be up to and including the maximum cap value (e.g., a deposit fee of 20% would be possible). |
| **Recommendation** | For consistency, consider using <= instead of < in the `require` statements for these values. |
| **Resolution** | RESOLVED<br><br>The client has implemented the recommendation. |

## 2.5 Timelock

The timelock is a fork of Compound Finance's timelock with a modification that allows the admin to set the delay directly, instead of requiring it be a queued transaction executed by the timelock itself.

| Parameter | Value | Description |
|---|---|---|
| **Delay** | 6 hours | 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.<br><br>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

| Issue #32 | Admin can update `delay` at any time |
|---|---|
| **Severity** | MEDIUM SEVERITY |
| **Location** | Lines 276-277<br><br>`function setDelay(uint delay_) public {`<br>`  require(msg.sender == admin, "Timelock::setDelay: Call must come from admin.");` |
| **Description** | The `setDelay` function allows the admin user to call it directly, instead of requiring a transaction queued on the timelock to perform this. This means that the effective delay on any transaction is really just the minimum delay, which is 6 hours, because the admin can set the delay to the minimum, then queue a transaction to be executed. |
| **Recommendation** | Only allow the timelock itself to update the delay. This gives users trust that the delay specified on the timelock will be respected for any transaction, and any change in delay can be seen ahead of time as a queued transaction.<br><br>`function setDelay(uint256 delay_) public {`<br>`  require(msg.sender == address(this), "Timelock::setDelay: Call must come from Timelock.");` |
| **Resolution** | ACKNOWLEDGED |


PALADIN
BLOCKCHAIN SECURITY