YieldYak Audit

MasterChef Strategies

March 2022

By CoinFabrik

Introduction

CoinFabrik was asked to audit some contracts for the YieldYak project. First we will provide a summary of our discoveries and then we will show the details of our findings.

Scope

The contracts audited are from the https://github.com/yieldyak/smart-contracts/ git repository. The audit is based on the commit `c3962a4530894e41bff0a8ef9c725b66f674c4bf`.

The audited files are:

- `contracts/strategies/MasterChefStrategy.sol`: Adapter strategy for MasterChef.
- `contracts/strategies/MasterChefStrategyForLP.sol`: Adapter strategy for MasterChef with LP deposit.
- `contracts/strategies/JoeStrategyForLP.sol`: Strategy for Joe's liquidity pools. It derives from `MasterChefStrategyForLP`.
- `contracts/strategies/MasterChefStrategyForSA.sol`: Adapter strategy for MasterChef with single-sided token deposit.

The scope of the audit is limited to those files. No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. In particular, it must be noted that `contracts/YakStrategyV2.sol` and `contracts/lib/DexLibrary.sol` were reviewed in a different audit. Also, no tests were reviewed for this audit.

Analyses

Without being limited to them, the audit process included the following analyses:

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers

- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

Summary of Findings

We found no critical or medium issues. Several minor issues were found. Also, several enhancements were proposed.

Security Issues

ID	Title	Severity	Status
MI-01	Missing Validation	Minor	Unresolved
MI-02	Solidity Compiler Version	Minor	Unresolved
MI-03	Only EOA Check Bypass	Minor	Unresolved
MI-04	Possible Reentrancy Problems	Minor	Unresolved
MI-05	WAVAX is Not Always the Reward Token	Minor	Unresolved
MI-06	On Wrong Deposit Fees	Minor	Unresolved
MI-07	On Wrong Withdrawal Fees	Minor	Unresolved

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

MasterChefStrategy

It inherits all the roles and permissions defined in the `YakStrategyV2` contract.

Owner

Besides all the functionality defined in the `YakStrategyV2` contract the contract owner can rescue the funds deployed by the strategy.

Dev

Besides all the functionality defined in the `YakStrategyV2` contract the contract dev can set the extra-reward swap-pair, used to transform extra-reward tokens into reward tokens.

It is also worth mentioning that by default the Dev address is `0x2D580F9CF2fB2D09BC411532988F2aFdA4E7BefF` in the contracts derived from `MasterChefStrategy`.

EOA

Besides all the functionality defined in the `YakStrategyV2` contract, an EOA can trigger a reinvest cycle.

MasterChefStrategyForLP

It inherits all roles in `MasterChefStrategy`. No new roles or capabilities are added.

MasterChefStrategyForSA

It inherits all roles in `MasterChefStrategy`. No new roles or capabilities are added.

JoeStrategyForLP

It inherits all roles in `MasterChefStrategy`.

The only added capability is that the staking contract, the rewarder, the dev and the owner can transfer AVAX to the contract.

Security Issues Found

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of but can be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit can have four distinct statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code but is a result of an intentional decision.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk

Critical Severity Issues

No critical issues were found.

Medium Severity Issues

No medium issues were found.

Minor Severity Issues

MI-01 Missing Validation

Location:

- `contracts/strategies/MasterChefStrategy.sol:103`

The code does not check that the rewardToken is `token1()` in the else path, which may lead to use a pair that converts to a non-desired ERC20 token as the `swapPairExtraReward`.

Recommendation

Use `DexLibrary.checkSwapPairCompatibility()` to validate that any pair that will be used by `DexLibrary.swap()` corresponds to the tokens intended in the swap.

Status

Unresolved.

MI-02 Solidity Compiler Version

All the audited files use the pragma `solidity 0.7.3;` statement. This implies that an old solidity version is being used which may lead into hitting already fixed bugs.

Recommendation

It is better to lock to a specific compiler version (for example, `pragma solidity 0.8.12;`) and keep it up to date. Also, when updating to 0.8 take into account the new semantics for safe math operations.

Status

Unresolved.

MI-03 Only EOA Check Bypass

Location:

- `contracts/strategies/MasterChefStrategy.sol:152-154`

In `MasterChefStrategy`, the `reinvest()` method is guarded by the `onlyEOA` modifier, stopping a contract from reinvesting. But if the unclaimed rewards exceed `MAX_TOKENS_TO_DEPOSIT_WITHOUT_REINVEST` a contract may call `deposit()` and `withdraw()` in a single transaction and do a reinvestment while keeping all the deposit tokens involved. This operation may even be made using funds obtained with a flash loan.

Recommendation

Decouple reinvesting from depositing tokens, do not check if the actor doing the reinvest is an EOA for consistency or both.

Status

Unresolved.

MI-04 Possible Reentrancy Problems

Location:

- `contracts/strategies/MasterChefStrategy.sol`

Several of the external (and public) functions in the `MasterChefStrategy` contract do several calls to other contracts and also invoke methods to be defined in derived contracts that should also invoke other contracts, leaving the possibility of being attacked with a reentrancy attack, which in the worst scenario may eventually lead to stolen funds.

Recommendation

Make the functions `deposit()`, `depositWithPermit()`, `depositFor()`, `withdraw()` and `reinvest()` non-reentrant.

Status

Unresolved.

MI-05 WAVAX is Not Always the Reward Token

Location:

- `contracts/strategies/MasterChefStrategy.sol:231-235`

If no `swapPairExtraReward` is set in a contract derived from `MasterChefStrategy` and it receives AVAX, it will lead to incorrect calculations based on incorrect information returned by the `_convertExtraTokensIntoReward()` function if the `rewardToken` is not WAVAX.

Recommendation

Either check that the reward token is WAVAX in the `_convertExtraTokensIntoReward()` function or do the enhancement proposed in EN-04, which should make this kind of issue impossible.

Status

Unresolved.

MI-06 On Wrong Deposit Fees

Location:

- `contracts/strategies/MasterChefStrategy.sol:160-164`

When depositing in contracts derived from `MasterChefStrategy`, the tokens staked after interacting with the master chef are calculated instead of measured in the `_deposit()` function. This means that if for any reason the amount of tokens staked are different from the calculated value the contract would be left in an inconsistent state.

Recommendation

Measure the tokens staked instead of calculating them in order to keep the internal state consistent. As this measurement will change depending on the derived contract, an abstract function that does it should be defined and used in the `MasterChefStrategy` contract and its children should define it.

If this recommendation is made, then the deposit fees are not necessary.

Status

Unresolved.

MI-07 On Wrong Withdrawal Fees

Location:

- `contracts/strategies/MasterChefStrategy.sol:168-178`

When doing a withdrawal in contracts derived from `MasterChefStrategy`, the tokens obtained after interacting with the master chef are calculated instead of measured in the `withdraw()` function. This means that if for any reason the tokens obtained are different from the calculated value the contract would be left in an inconsistent state.

Recommendation

Measure the tokens obtained instead of calculating them in order to keep the internal state consistent. If you do so, then the withdrawal fees are not necessary.

Status

Unresolved.

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

Table

ID	Title	Status
EN-01	Another Swap-Pairs Check In DexLibrary	Not implemented
EN-02	Extra Gas Usage in Deposit	Not implemented
EN-03	More Robust Reinvest Procedure	Not implemented
EN-04	When WAVAX is the Reward Token	Not implemented

Details

EN-01 Another Swap-Pairs Check In DexLibrary

We noticed that the logic to validate that the `swapPairToken0` and `swapPairToken1` are correct in order to invoke the `DexLibrary.convertRewardTokensToDepositTokens()` function is repeated in a lot of your contracts, including `MasterChefStrategyForLP`.

Recommendation

Add an extra function in `DexLibrary` to check that `rewardToken` and `depositToken` are compatible with `swapPairToken0` and `swapPairToken1`. This new function would be similar in intent to `DexLibrary.checkSwapPairCompatibility()`, which checks that the pair is and addresses are correct to invoke `DexLibrary.swap()`.

Use this new function to simplify the `MasterChefStrategyForLP.assignSwapPairSafely()` function, and the validation logic for the rest of the strategies that use `DexLibrary.convertRewardTokensToDepositTokens()`.

Status

Not implemented.

EN-02 Extra Gas Usage in Deposit

Location:

- `contracts/strategies/MasterChefStrategy.sol:145-155`

When interacting with a contract derived from `MasterChefStrategy` a depositor may spend extra gas because a reinvest operation is forced when doing a deposit if there are enough unclaimed rewards.

Recommendation

Decouple reinvest from deposit. The incentives to do the reinvest should be enough to keep the reinvest procedure running.

Status

Not implemented.

EN-03 More Robust Reinvest Procedure

Location:

- `contracts/strategies/MasterChefStrategy.sol:249-267`

Instead of converting the pool-reward tokens and the extra-reward tokens into reward tokens, looking at the total balance, and then converting them into deposit tokens to stake them, the `MasterChefStrategy._reinvest()` method uses an estimation of the reward token balance to pay fees (both dev fee and reinvest rewards) and stake deposit tokens. This may lead to either reward tokens not used in a reinvest iteration or attempting to convert more tokens than the belongings of the contract, leading to a transaction reversion.

Recommendation

Instead of trusting the values returned by the `_convertPoolTokensIntoReward()` and `_convertExtraTokensIntoReward()` functions, do the conversion and use the resulting balance to calculate fees and do the staking. This should make the code more robust. Note that no extra gas is required as the reward token balance should only be calculated after converting the rest of the tokens.

Status

Not Implemented.

EN-04 When WAVAX is the Reward Token

Location:

- `contracts/strategies/MasterChefStrategy.sol:231-235,323`

When WAVAX is the reward token, the MasterChefStrategy contract has extra logic to handle the AVAX-WAVAX conversion. This logic is intermingled with the rest of the code making it more difficult to understand.

Recommendation

If only WAVAX is used as the `rewardToken` then make it the only possibility. If both WAVAX and other tokens need to be supported as the `rewardToken` in the `MasterChefStrategy` contract, then make 2 derived contracts. One should handle only WAVAX as its `rewardToken` and the other should not attempt to transform AVAX into WAVAX.

Status

Not Implemented.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest for other stakeholders of the project, including users of the audited contracts, owners or project investors.

Centralization

The dev and the owner of each of the analyzed contracts have privileged capabilities.

The development team informed us that they are using a community multisig to help mitigate centralization risks.

Upgradeability

No provisions to upgrade the contracts are found on any of the analyzed contracts.

Tests

Tests for this project are developed in a private git repository. The development team informed us that they have tests for, among others, `JoeStrategyForLP` and `GmxStrategyForGMX`. Those contracts derive from all the abstract contracts analyzed in this audit.

Changelog

- 2022-03-16 – Initial report based on commit `c3962a4530894e41bff0a8ef9c725b66f674c4bf`.
- 2022-03-17 – Split ME-01 into MI-06 and MI-07 and lower its severity.