YieldYak Audit

YakToken and DEX Strategies

November 2021

By CoinFabrik

Introduction

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

Summary

The contracts audited are from the “Smart Contracts” and “Governance Contracts” repositories at https://github.com/yieldyak/smart-contracts/ and https://github.com/yieldyak/governance-contracts/ respectively.

For the "Smart Contracts" repository, the audit is based on commit 8f173a4cfd3302db0f888c0af33a1fb1e6f47677.

For the "Governance Contracts" repository, the audit is based on commit 86e79fa2ab58115ea42072632c07307ab52acda5.

Contracts

The audited contracts are:

- "Smart Contracts" repository:
  - `contracts/strategies/JoeStrategyV3.sol`: Strategy for Trader Joe, which includes optional and variable extra rewards.
  - `contracts/strategies/DexStrategyV6.sol`: Strategy for StakingRewards.
  - `contracts/timelocks/YakTimelockForDexStrategyV3.sol`: Generic timelock for YakStrategies
- "Governance Contracts" repository:
  - `contracts/YakToken.sol`: The governance token for Yield Yak. ERC-20 with fixed supply + offchain signing.

Analyses

The following analyses were performed:

- Misuse of the different call methods

- Integer overflow errors
- Division by zero errors
- Outdated version of Solidity compiler
- Front running attacks
- Reentrancy attacks
- Misuse of block timestamps
- Softlock denial of service attacks
- Functions with excessive gas cost
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Failure to use a withdrawal pattern
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures

Findings and Fixes

ID	Title	Severity	Status
CR-01	Malign Staking Contract May Steal All Deposit Tokens	Critical	Not fixed
CR-02	Moving Funds Away From Staking Contracts Breaks Strategies	Critical	Not fixed
ME-01	Denial of Service While Reinvesting	Medium	Not fixed
ME-02	Wrong Permissions in JoeStrategyV3.setExtraRewardSwapPair()	Medium	Not fixed
MI-01	Solidity Compiler Version	Minor	Not fixed
MI-02	Incomplete Swap-Pair Checks in DexStrategyV6.assignSwapPairSafely()	Minor	Not fixed
MI-03	Unnecessary Reentrancy in withdraw(), _deposit() and _reinvest()	Minor	Not fixed
MI-04	Only EOA Check Bypass	Minor	Not fixed
EN-01	Automated Testing	Enhancement	Not fixed
EN-02	Gas Optimization in YakToken.getDomainSeparator()	Enhancement	Not fixed
EN-03	Wrong Error Message on _safeTransfer()	Enhancement	Not fixed
EN-04	Wrong Contract Name	Enhancement	Not fixed
EN-05	Repeated Calculations in JoeStrategyV3	Enhancement	Not fixed

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**.
- **Enhancement:** These kinds of findings do not represent a security risk. They are best practices that we suggest to implement.

This classification is summarized in the following table:

SEVERITY	EXPLOITABLE	ROADBLOCK	TO BE FIXED
Critical	Yes	Yes	Immediately
Medium	In the near future	Yes	As soon as possible
Minor	Unlikely	No	Eventually
Enhancement	No	No	Eventually

Issues Found by Severity

Critical severity

CR-01 Malign Staking Contract May Steal All Deposited Tokens

The `setAllowances()` function, invoked in the constructors of both `DexStrategyV6` and `JoeStrategyV3` gives infinite allowance to the staking contract. See line 80 of `DexStrategyV6.sol` and line 102 of `JoeStrategyV3.sol`. This may allow a malign staking contract to steal all the deposited tokens, even after running `emergencyWithdraw()` or `rescueDeployedFunds()`.

Recommendation

Do not give an infinite allowance. Instead, give the minimum required allowance when staking tokens (line 172 of `DexStrategyV6.sol` and line 227 of `JoeStrategy.sol`).

CR-02 Moving Funds Away From Staking Contracts Breaks Strategies

After `DexStrategyV6.rescueDeployedFunds()`, `JoeStrategyV3.emergencyWithdraw()` or `JoeStrategyV3.rescueDeployedFunds()` are executed the `DexStrategyV6.withdraw()` and `JoeStrategyV3.withdraw()` functions revert when a user tries to withdraw funds.

It is even worse if some other user deposits tokens afterwards. Then some users will be able to withdraw but others will not, depending on the funds available in the staking contract.

Recommendation

The `DexStrategyV6.withdraw()` and `JoeStrategyV3.withdraw()` functions must handle that the deposit tokens may be owned by either the staking contract or the strategy.

An alternative approach for `JoeStrategyV3.emergencyWithdraw()` is to disable the contract and manually return the funds to their owners using the YakStrategy.recover*() functions that are available to the strategy given that it inherits from `YakStrategy`.

Another option is to remove the `DexStrategyV6.rescueDeployedFunds()`, `JoeStrategyV3.emergencyWithdraw()` and `JoeStrategyV3.rescueDeployedFunds()` functions.

Medium severity

ME-01 Denial of Service While Reinvesting

If the contract developer, the contract owner or the address doing the transaction where the `_reinvest()` function is being executed rejects the payment for doing the reinvestment, the reinvestment is rejected. It happens in both `DexStrategyV6` and `JoeStrategyV3`.

This manifests itself in the `_reinvest()` function. In `DexStrategyV6` look at lines 143, 148 and 153. In `JoeStrategyV3` look at lines 198, 203 and 208.

The developer may block the reinvestment process permanently.

This denial of service may happen while invoking the following public functions:

- `DexStrategyV6.reinvest()`
- `DexStrategyV6.deposit()`
- `DexStrategyV6.depositFor()`
- `DexStrategyV6.depositWithPermit()`
- `JoeStrategyV3.reinvest()`
- `JoeStrategyV3.deposit()`
- `JoeStrategyV3.depositFor()`
- `JoeStrategyV3.depositWithPermit()`

Recommendation

Use the "Favor pull over push for external calls" pattern to transfer fees, isolating each transfer in its own transaction. See https://eth.wiki/howto/smart-contract-safety#favor-pull-over-push-for-external-calls for details.

ME-02 Wrong Permissions in JoeStrategyV3.setExtraRewardSwapPair()

The function `JoeStrategyV3.setExtraRewardSwapPair()` (defined in line 241 of `JoeStrategyV3.sol`) can be executed by the strategy developer. This allows the developer to change the pair, and set a potentially harmful contract as swap-pair after deployment of the strategy, potentially stealing all extra rewards.

This issue severity has been reduced because the developer will usually be trustworthy.

Recommendation

Make the `JoeStrategyV3.setExtraRewardSwapPair()` `onlyOwner` instead of `onlyDev` or remove the option to set the extra swap pair after the construction of the strategy, keeping consistency with the rest of the swap pairs.

Minor severity

MI-01 Solidity Compiler Version

All audited files use the pragma `solidity ^0.7.0;` statement. This implies that an old solidity version is being used and also adds risks because bugs may be introduced by using a different solidity compiler. See https://swcregistry.io/docs/SWC-103.

Recommendation

It is better to lock to a specific compiler version (for example, `pragma solidity 0.8.10;`) and keep it up to date. Also, when updating to 0.8 take into account the new semantics for safe math operations.

MI-02 Incomplete Swap-Pair Checks in DexStrategyV6.assignSwapPairSafely()

In the `Dex6StrategyV6` contract, the pairs used are not properly controlled.

The association between the reward token and the `_swapPairToken1` is not being checked.

Token association has not been checked properly when the deposit token is associated with any over the swap pair tokens.

Recommendation

Check all the pair associations. Use `DexLibrary.checkSwapPairCompatibility()` to simplify the `DexStrategyV6.assignSwapPairSafely()` code.

MI-03 Unnecessary Reentrancy in withdraw(), _deposit() and _reinvest()

The functions `DexStrategyV6._deposit()` (line 104), `DexStrategyV6.withdraw()` (line 114-115), `DexStrategyV6._reinvest()` (line164), `JoeStrategyV3._deposit()` (line 127), `JoeStrategyV3.withdraw()` (lines 137-138) and `JoeStrategyV3._reinvest()` (line 164) allow reentrancy when calling arbitrary code in the blockchain. This may be used to retrieve more funds than expected.

Recommendation

For `DexStrategyV6.withdraw()` and `JoeStrategyV3.withdraw()` add a reentrancy guard. For the rest of the functions add a reentrancy guard or move the referred lines just before the emit statement, to follow the CEI pattern. See

https://docs.soliditylang.org/en/v0.8.10/security-considerations.html#use-the-checks-effects-interactions-pattern for details.

MI-04 Only EOA Check Bypass

In both `DexStrategyV6` and `JoeStrategyV3`, 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.

Recommendation

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

Enhancements

EN-01 Automated Testing

None of the contracts in the smart-contracts repo that we audited has automated tests.

Recommendation

Add automated tests with proper coverage for those contracts.

EN-02 Gas Optimization in YakToken.getDomainSeparator()

The function `YakToken.getDomainSeparator()` (lines 179-189 of `YakToken.sol`) can be optimized in 2 ways:

- Given that `name` is constant, do not calculate `keccak256(bytes(name))` on each invocation.
- Given that `_getChainId()` will almost always return the same value, cache the calculated value to avoid repeating the calculation when it does not change.

EN-03 Wrong Error Message on _safeTransfer()

The error message is inconsistent with the rest of the error messages in the contracts when a transfer fails in the `_safeTransfer()` function in both `DexStrategyV6` and `JoeStrategyV3`.

Recommendation

Use `DexStrategyV6::_safeTransfer` and `JoeStrategyV3::_safeTransfer` as error messages respectively for consistency.

EN-04 Wrong Contract Name

The name of the contract in the `JoeStrategyV3.sol` file is `JoeStrategyV2`.

Recommendation

Rename the contract to `JoeStrategyV3`.

EN-05 Repeated Calculations in JoeStrategyV3

When executing the functions `JoeStrategyV3._deposit()` (lines 118-132 of `JoeStrategyV3.sol`) and `JoeStrategyV3.reinvest()` (lines 150-155 of `JoeStrategyV3.sol`) the calculations made in the function `_checkReward()` is

made twice, once directly inside the function and the second one in the `checkReward()` (without "_") invocation. This generates extra gas usage.

Recommendation

Refactor these functions to do the `_checkReward()` calculations only once.

Other considerations

Non audited code

`JoeStrategyV3` and `DexStrategyV6` make heavy use of other contracts and libraries included in the same repository. Those were not reviewed in this audit, because they were out of scope, and we were informed by the development team that they were not audited in a different audit either.

Centralization

The owner of the `JoeStrategyV3` and `DexStrategyV6` may withdraw all funds from the staking contract and then out of the strategy. The extraction of the funds out of the strategy contract is implemented in the base contract `YakStrategy.recover*()` functions. This functionality exists to be able to recover if there is a problem with the strategy.

A user of the strategies needs to trust the strategy owner because of this.

Documentation

The libraries and base contract used to implement the trading strategies are not documented. In order to compensate for this, we looked at the code and the development team gave us an informal description of them.

Extra gas usage in deposit

In both `JoeStrategyV3` and `DexStrategyV6` a depositor may spend extra gas because a reinvest operation is forced when doing a deposit if there are enough unclaimed rewards.

This is the offending code in `DexStrategyV6._deposit()` (lines 98-103 of `DexStrategy.sol`):

```
if (MAX_TOKENS_TO_DEPOSIT_WITHOUT_REINVEST > 0) {
    uint unclaimedRewards = checkReward();
    if (unclaimedRewards > MAX_TOKENS_TO_DEPOSIT_WITHOUT_REINVEST) {
        _reinvest(unclaimedRewards);
    }
}
```

And this is the offending code in `JoeStrategyV3._deposit()` (lines 120-126 of `JoeStrategyV3.sol`):

```
if (MAX_TOKENS_TO_DEPOSIT_WITHOUT_REINVEST > 0) {
    uint unclaimedRewards = checkReward();
    if (unclaimedRewards > MAX_TOKENS_TO_DEPOSIT_WITHOUT_REINVEST) {
        (uint poolTokenAmount, address extraRewardTokenAddress, uint
extraRewardTokenAmount, uint rewardTokenAmount) = _checkReward();
        _reinvest(poolTokenAmount, extraRewardTokenAddress, extraRewardTokenAmount,
rewardTokenAmount);
    }
}
```

Conclusion

We have found two critical issues, two medium issues and several minor issues. Also several enhancements were proposed.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the YieldYak project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.