YieldYak Audit

AaveStrategyAvaxV1, DEXLibrary and dependencies

December 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.

Scope

The contracts audited are from the https://github.com/yieldyak/smart-contracts/ git repository. The audit is based on the commit `5dafbd86cc0368c5ebabb04db560df49ab202c7d`.

The audited contracts are:

- `AaveStrategyAvaxV1.sol`: Yield-farming strategy for WAVAX Aave lending pools. The general idea for the strategy is documented in https://yieldyak.medium.com/aave-strategies-d36d95b15b61.
- `DexLibrary.sol`: Library used to interact with distributed exchanges.
- YakStrategyV2.sol: Base contract for yield-farming strategies.
- `YakERC20.sol:` Base contract to make an ERC20 token. Used to track deposits in yield-farming strategies.
- `YakStrategyV2Payable.sol:` Base contract for yield-farming strategies where AVAX is deposited instead of an ERC20 token.
- `Permissioned.sol:` Utility base contract used to restrict who can deposit in a yield-farming strategy.

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- Outdated version of Solidity compiler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 Token Delegator May Steal All Tokens and AVAX in AaveStrategyAvaxV1	Critical	Not fixed
ME-01	Overflow in YakStrategyV2 while mapping deposit tokens to shares	Medium	Not fixed
MI-01	Solidity Compiler Version	Minor	Not fixed
MI-02	DexLibrary swap fee documentation	Minor	Not fixed
MI-03	Overflows in DexLibrary	Minor	Not fixed
MI-04	Divisions by Zero in DexLibrary	Minor	Not fixed
MI-05	Owner May Block Reinvest Process	Minor	Not fixed
MI-06	Only EOA Check Bypass	Minor	Not fixed
EN-01	Exception message errors	Enhancement	Not fixed
EN-02	Test suites	Enhancement	Not fixed
EN-03	Misleading Strategy Parameter Name in AaveStrategyAvaxV1	Enhancement	Not fixed
EN-04	Another Misleading Strategy Parameter Name in AaveStrategyAvaxV1	Enhancement	Not fixed
EN-05	Overflow in YakStrategyV2.getSharesForDepositTokens()	Enhancement	Not fixed
EN-06	Overflow in YakStrategyV2.getDepositTokensForShares()	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 Issues

CR-01 Malign Token Delegator May Steal All Tokens and AVAX in AaveStrategyAvaxV1

The `setAllowances()` function, invoked in the constructor of `AaveStrategyAvaxV1,` gives infinite allowance to the token delegator. See lines 123-124 of `AaveStrategyAvaxV1.sol`. This may allow a malign token delegator to steal all the strategy tokens, even after running `rescueDeployedFunds().`

Recommendation

Do not give an infinite allowance. Instead, give the minimum required allowance in the transaction when an actual transfer is being made.

Medium Severity Issues

ME-01 Overflow in YakStrategyV2 while mapping deposit tokens to shares

An integer overflow may occur while running `YakStrategyV2.getSharesForDepositTokens()` (line 156) and `YakStrategyV2.getDepositTokensForShares()` (line 168) if `totalSupply * totalDeposits() >=` 2^{256}. The offending and repeating code is

```
if (totalSupply.mul(totalDeposits()) == 0) {
```

Given that `YakStrategyV2.getSharesForDepositTokens()` should be used in the deposit process and `YakStrategyV2.getDepositTokensForShares()` should be used in the withdrawal process this issue may effectively block them both. And given that the overflow is generated using only persistent state, the block may be permanent.

Recommendation

Check individually for zero `totalSupply` and `totalDeposits()`.

```
if(totalSupply == 0 || totalDeposits() == 0){
```

Minor Severity Issues

MI-01 Solidity Compiler Version

All audited files use the pragma `solidity 0.7.3;` 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 DexLibrary swap fee documentation

The `getAmountOut()` function (lines 120-125 of `DexLibrary.sol`) has documented that it assumes a 0.3% swap fee but the functions that use it (`swap()` and `estimateConversionThroughPair()`) do not.

Recommendation

Either document the unique swap fee on every function, add it as a parameter, have a registry with the swap fee for each IPair or do a combination of the suggested solutions.

MI-03 Overflows in DexLibrary

The `_quoteLiquidityAmountOut()` (in lines 97-99 of `Dexlibrary.sol`) and `getAmountOut()` (in lines 120-125 of `DexLibrary.sol`) may overflow if big numbers are passed as parameters.

Recommendation

Document safe parameter values for the mentioned functions and its dependencies in `DexLibrary.sol`. Adding a require statement is a possible solution but it has to be clear to the user of the library, who will likely take measures to avoid being in the situation, because a denial of service may be triggered. Also check the current uses of the library for possible overflows.

MI-04 Divisions by Zero in DexLibrary

The `DexLibray.getAmountOut()` function will raise a divide by zero exception in line 123 of `DexLibrary.sol` if `reserveIn` is zero.

The `DexLibrary._quoteLiquidityAmountOut()` function will raise a divide by zero exception in line 98 of `DexLibrary.sol` if `reserve0` is zero.

Recommendation

Add a `require` statement to each function checking that the offending variable is not zero. Also document the requirement on other functions in DexLibrary that use those.

MI-05 Owner May Block Reinvest Process

If the sum of `DEV_FEE_BIPS`, `ADMIN_FEE_BIPS` and `REINVEST_REWARD_BIPS` is bigger than `BIPS_DIVISOR` then the reinvest process will revert when it runs out of funds while transferring fees. This is triggered in the `AaveStrategyAvaxV1._reinvest() function`. This will also apply to other strategies that inherit from YakStrategyV2.

Recommendation

Add `REINVEST_REWARDS_BIPS` in the require expression of lines 197, 207, and 217 of `YakStrategyV2.sol`.

MI-06 Only EOA Check Bypass

In `AaveStrategyAvaxV1`, 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 Exception message errors

1. In line 135 of `DexLibrary.sol` the error says `"DexLibrary::TRANSFER_FROM_FAILED"` but a normal transfer is being made.
2. In line 307 of `AaveStrategyAvaxV1.sol` the error says `"AaveStrategyAvaxV1::TRANSFER_FROM_FAILED"` but a normal transfer is being made.
3. In line 52 of `DexLibrary.sol` the error message says `"DexLibrary::_convertRewardTokensToDepositTokens"` but the method where the error is raised does not start with an underscore.
4. In line 48 of `YakStrategyV2.sol` the error says `"YakStrategy::onlyEOA"` but the contract name is `YakStrategyV2`.
5. In line 56 of `YakStrategyV2.sol` the error says `"YakStrategy::onlyDev"` but the contract name is `YakStrategyV2`.

EN-02 Test suites

The client told us that they have test suites for several strategies in another code repository. Three of those test-suites, corresponding to `DexStrategyV6`, `JoeStrategyV4` and `AaveStrategyAvaxV1` were shared with us by the development team via an informal channel.

Recommendation

Have the code and test suite in the same repository.

EN-03 Misleading Strategy Parameter Name in AaveStrategyAvaxV1

The `minMinting` parameter name, used in lines 170 and 245 of `AaveStrategyAvaxV1.sol` is misleading because it is not involved in any minting.

Recommendation

Rename the `minMinting` parameter to `minWithdrawWAVAX` (or similar).

EN-04 Another Misleading Strategy Parameter Name in AaveStrategyAvaxV1

The `MAX_TOKENS_TO_DEPOSIT_WITHOUT_REINVEST` parameter name is also misleading because it is used in line 159 of `AaveStrategyAvaxV1.sol` to compare with the accumulated rewards (instead of the deposited tokens),

Recommendation

Rename the `MAX_TOKENS_TO_DEPOSIT_WITHOUT_REINVEST` parameter to `MAX_REWARDS_WITHOUT_REINVEST_ON_DEPOSIT` (or similar).

EN-05 Overflow in YakStrategyV2.getSharesForDepositTokens()

The `YakStrategyV2.getSharesForDepositTokens()` function will overflow in line 159 of `YakStrategyV2.sol` if `amount * totalSupply >=` 2^{256}.

This is the offending line:

```
return amount.mul(totalSupply).div(totalDeposits());
```

Recommendation

Use the `mul()` function defined in line 108 of SafeMath.sol to give a better error message if the overflow occurs. A better error message is enough because if a deposit was stopped by the overflow it can still be made doing several deposits of a smaller amount. If you upgrade the codebase to solidity 0.8 or above you may want to use the `tryMul()` function available in OpenZeppelin's safe-math library.

```
return amount.mul(totalSupply, "YakStrategyV2:amount exceeded than allowed, try with a
smaller amount").div(totalDeposits());
```

EN-06 Overflow in YakStrategyV2.getDepositTokensForShares()

The `YakStrategyV2.getDepositTokensForShares()` function will overflow in line 171 of `YakStrategy.sol` if `amount * totalDeposits() >=` 2^{256}.

This is the offending line:

```
return amount.mul(totalDeposits()).div(totalSupply);
```

Recommendation

Use the `mul()` function defined in line 108 of `SafeMath.sol` to give a better error message if the overflow occurs. A better error message is enough because if a withdrawal was stopped by the overflow it can still be made doing several withdraws of a smaller amount. If you upgrade the codebase to solidity 0.8 or above you may want to use the `tryMul()` function available in OpenZeppelin's safe-math library.

Other Considerations

Centralization

The owner of `AaveStrategyAvaxV1` may withdraw all funds from the token delegator and then out of the strategy. The extraction of the funds out of the strategy contract is implemented in the base contract `YakStrategyV2.recover*()` functions and in the `AaveStrategyAvaxV1.rescueDeployedFunds()` function. This functionality exists to be able to recover if there is a problem with the strategy.

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

Extra gas usage in deposit

In `AaveStrategyAvaxV1` 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 `AaveStrategyAvaxV1._deposit()` (lines 157-162 of `DexStrategy.sol`):

```
if (MAX_TOKENS_TO_DEPOSIT_WITHOUT_REINVEST > 0) {
    uint256 avaxRewards = _checkRewards();
    if (avaxRewards > MAX_TOKENS_TO_DEPOSIT_WITHOUT_REINVEST) {
        _reinvest(avaxRewards);
    }
}
```

Conclusion

We have found one critical issue, one medium issue 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.