YieldYak Audit

YakStrategyManagerV1

January 2022

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 `de8a03b5c5bc67166f4ba167751655bf410b46ed`. Fixes where checked in commit `8fbcedfbc7d2350863017dc452c3a42b1d3c9266`.

The audited contracts are:

- `contracts/timelocks/YakStrategyManagerV1.sol`: Role-based manager for YakStrategy contracts
- `contracts/lib/SafeMath.sol`: Library used to avoid silent underflows and overflows on integer operations.

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 `contracts/lib/AccessControl.sol`, `contracts/lib/EnumerableSet.sol`, `contracts/lib/Address.sol` and `contracts/lib/Context.sol` were taken from the OpenZeppelin contracts library (version 3.4.2) and then only changed pragmas and import paths. 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- Outdated version of Solidity compiler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 one minor issue which was fixed by the development team. Also, one enhancement was proposed and it will be made in a future release.

Security Issues

ID	Title	Severity	Status
MI-01	Inconsistent Roles	Minor	Resolved

Privileged Roles

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

YakStrategyManagerV1

Roles in this contract are based upon the machinery built in the `AccessControl` abstract contract.

Default Admin

Addresses with this role can administer which addresses have which roles in the system, by calling the public methods defined in the `AccessControl` base contract. By default, the manager (passed in the constructor) has this role.

Strategy Owner Setter

Addresses with this role can initiate the process to give away the ownership of a managed strategy by calling the `proposeOwner()` function. This operation is time locked and will be effectively made only after 14 days by invoking the `setOwner()` function. In the meantime, it can be reverted by calling `proposeOwner()` again, passing the address of this contract as `newOwner`.

By default, only the manager (passed in the constructor) has this role.

Emergency Rescuer

Addresses with this role can make this contract call the `revokeAllowance()` and `rescueDeployerFunds()` functions in the managed strategies. By default, only the team (passed in the constructor) has this role.

Emergency Sweeper

Addresses with this role can:

- Make this contract call the `recoverERC20()` method in the managed strategies.
- Transfer ERC20 tokens owned by the contract to themselves.
- Make this contract call the `recoverAVAX()` method in the managed strategies.
- Transfer AVAX owned by the contract to themselves.

By default, only the deployer (passed in the constructor) has this role.

Global Max Fee Setter

Addresses with this role can alter the global max fee that can be set in a strategy as a percentage of the invested funds. By default, only the team (passed in the constructor) has this role.

Fee Setter

Addresses with this role can:

- Make this contract alter the fees by invoking the `updateAdminFee()`, `updateDevFee()` and `updateReinvestReward()` functions in a managed strategy, respecting the global max fee.
- Make this contract call the `updateMinTokensToReinvest()` function in a managed strategy.
- Make this contract call the `updateMaxTokensToDepositWithoutReinvest()` function in a managed strategy.

By default, the deployer and the team (passed in the constructor) have this role.

Strategy Permissioner

Addresses with this role can:

- Make this contract call the `allowDepositor()` function in a managed strategy.
- Make this contract call the `removeDepositor()` function in a managed strategy.

By default, only the team (passed in the constructor) has this role.

Strategy Disabler

Addresses with this role can:

- Make this contract call the `revokeAllowance()` function in a managed strategy.
- Make this contract call the `updateDepositsEnabled()` function with false as a parameter in a managed strategy.

By default, the team and the deployer (passed in the constructor) have this role.

Strategy Enabler

Addresses with this role can:

- Make this contract call the `setAllowances()` function in a managed strategy,
- Make this contract call the `updateDepositsEnabled()` function with true as a parameter in a managed strategy.

By default, the team (passed in the constructor) has this role.

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.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk

Critical Severity Issues

No issues found.

Medium Severity Issues

No issues found.

Minor Severity Issues

MI-01 Inconsistent Roles

Location:

- `contracts/timelocks/YakStrategyManagerV1.sol:172,178`

The `revokeAllowance()` function can be executed by a strategy disabler or an emergency rescuer, but the documentation states that only a strategy disabler can do it.

Recommendation

Make the documentation and code consistent.

Status

Resolved. Docstring was fixed in commit `8fbcedfbc7d23508630l7dc452c3a42b1d3c9266`.

Enhancements

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

Table

ID	Title	Status
EN-01	Timelock to move funds	Not implemented

Details

EN-01 Timelock to move funds

Location:

- `contracts/timelocks/YakStrategyManagerV1.sol:288,300,342`

In order to be able to recover from eventual hacks and be able to recover lost funds, add a timelock to transfer AVAX or an ERC20 out of this contract and a method to cancel the transfer in the timelock window.

Status

Not implemented. The development team acknowledged the enhancement and informed us that it will be addressed in a future version.

Other Considerations

Centralization

The `YakStrategyManagerV1` reason of being is to give different addresses the ability to manage an investment strategy in different ways. Each one has different ways of making the strategy user lose money, so they have to be trusted. Three of those roles are particularly important.

A strategy owner setter may lead to the strategy to be completely owned by someone else. This risk is mitigated by the timelock used to do the actual ownership transfer.

An emergency sweeper may steal all the funds handled by each of the managed strategies.

A default admin may set any address to be either a strategy owner setter or an emergency sweeper (or any other of the contract roles).

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

Upgradeability

The `YakStrategyManagerV1` contract does not contain explicit provisions to do upgrades, but we consider they are not required because if an improved manager is developed and deployed, a strategy owner setter may transfer the ownership of the managed strategies to the new manager.

Changelog

- 2022-01-14 – Initial report based on commit `de8a03b5c5bc67166f4ba167751655bf410b46ed`.
- 2022-02-09 – Check fixes made for commit `8fbcedfbc7d2350863017dc452c3a42b1d3c9266`.