CoinFabrik

YieldYak Audit

YakRegistry, YakVault and YakStrategy

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 `2a7bc7e0a0ad71fcd60f5d53ab7ed8c350079b93`. Fixes were checked in commit `51f45e446c3fe95ee98679a68f184d8699b93fee`.

The audited files are:

- `contracts/YakRegistry.sol`: List of strategies officially supported by YieldYak.
- `contracts/YakVault.sol`: Managed deposit vault for *deposit tokens* that accepts deposits in the form of *deposit tokens* or *strategy tokens*. It must be noted that while the file is named `YakVault.sol`, the contract defined inside is named `YakVaultForSA`. For commit `51f45e446c3fe95ee98679a68f184d8699b93fee` this file has been renamed as `YakVaultForSA.sol` to match the name of the contract.
- `contracts/YakStrategy.sol`: Base contract for YieldYak investment strategies.
- `contracts/YakERC20.sol`: YRT token implementation.
- `contracts/interfaces/IERC20.sol`: Interface defining functions available in ERC20 tokens.
- `contracts/lib/SafeMath.sol`: Library used to avoid silent underflows and overflows on integer operations.
- `contracts/lib/Permissioned.sol`: Abstract contract that contains the logic to handle depositors.

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/Ownable.sol`, `contracts/lib/Context.sol`, `contracts/lib/EnumerableSet.sol`, `contracts/lib/SafeERC20.sol` and `contracts/lib/Address.sol` were taken from the OpenZeppelin contracts library (version 3.4.2) and then only changed pragmas, import paths and other minor and not relevant changes were made. 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- Outdated version of Solidity compiler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 1 critical issue, 3 medium issues and several minor issues. Also, one enhancement was proposed.

All critical and medium issues were resolved except for ME-02, which was mitigated. The minor issues were either resolved, partially resolved or acknowledged.

Security Issues

ID	Title	Severity	Status
CR-01	Missing Withdrawn Funds If YakVaultForSA Has Liquid Deposits	Critical	Resolved
ME-01	A Rogue Strategy May Steal Funds Handled by a Different Strategy in YakVaultForSA	Medium	Resolved
ME-02	YakVaultForSA Denial Of Service	Medium	Mitigated
ME-03	User Unable To Withdraw Funds	Medium	Resolved
MI-01	Solidity Compiler Version	Minor	Acknowledged
MI-02	Multiple Overflows in Shares/Tokens Conversions	Minor	Partially Resolved
MI-03	Unregistered Strategy is Enabled	Minor	Resolved
MI-04	Wrong documentation for YakVaultForSA.setActiveStrategy()	Minor	Resolved

Privileged Roles

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

YakRegistry

Owner

The owner can:

- Add a strategy.
- Pause a strategy.
- Disable a strategy.
- Resume a strategy.

By default, the owner is the address that deployed the contract.

YakVaultForSA

Owner

The owner can:

- Add and remove strategies using the `addStrategy()` and `removeStrategy()` functions.
- Set the active strategy using the `setActiveStrategy()` function.
- Withdraw vault-owned deposit tokens from any strategy, using the `withdrawFromStrategy()` and `withdrawPercentageFromStrategy()` functions. The tokens will be owned by the vault itself afterwards.
- Deposit deposit tokens for the vault owner into any strategy, using the `depositToStrategy()` and `depositPercentageToStrategy()` functions.

By default, the owner is the address that deployed the contract.

YakERC20

Approved Address

An address may allow a different address to transfer funds away from its belongings by either using the `approve()` function or signing off-line an approval to be passed to the `permit()` function.

Permissioned

Owner

The owner can add and remove allowed depositors.

By default, the owner is the address that deployed the contract.

Allowed Depositor

By default, there are no allowed depositors. And no operations are defined for an allowed depositor in this contract. Child contracts may define operations for them.

YakStrategy

Besides all the roles documented below, it also has the privileged roles documented for the `YakERC20` and `Permissioned` contracts. Also, given that `YakStrategy` is an abstract contract, strategies inherited from it will have other roles.

Owner

The owner can:

- Invoke the `setAllowances()` function (only documented, virtual function).
- Invoke the `revokeAllowance()` function.
- Invoke the `updateMinTokensToReinvest()` function.
- Invoke the `updateMaxTokensToDepositWithoutReinvest()` function.
- Invoke the `updateDevFee()` function.
- Invoke the `updateAdminFee()` function.
- Invoke the `updateReinvestReward()` function.
- Invoke the `updateDepositsEnabled()` function.
- Invoke the `recoverERC20()` function.
- Invoke the `recoverAVAX()` function.

By default, the owner is the address that deployed the contract.

Dev

The dev can invoke the `updateDevAddr()` function.

EOA

While no operations are defined for this role the `onlyEOA()` modifier is defined to aid in the creation of functions that can only be invoked by an EOA.

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 problem. 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

CR-01 Missing Withdrawn Funds If YakVaultForSA Has Liquid Deposits

Location:

- `contracts/YakVault.sol:96-122`

If the vault has liquid deposits, when a user withdraws its funds, less deposit tokens than expected are transferred to the user. The number of tokens not transferred corresponds to the liquid funds.

Recommendation

Fix `YakVaultForSA.withdraw()` logic.

Status

Resolved. Fixed for version `51f45e446c3fe95ee98679a68f184d8699b93fee`.

Medium Severity Issues

ME-01 A Rogue Strategy May Steal Funds Handled by a Different Strategy in YakVaultForSA

Location:

- `contracts/YakVault.sol:96-122`

A rogue strategy may steal funds handled by other strategies via a reentrancy attack using the `withdraw()` function.

Hereunder is the function code. We underlined all the opportunities for reentrancies.

```
function withdraw(uint256 amount) external {
   require(checkStrategies() == true, "YakVault::withdraw paused");
   uint256 depositTokenAmount = getDepositTokensForShares(amount);
   require(depositTokenAmount > 0, "YakVault::withdraw, amount too low");
   uint256 liquidDeposits = depositToken.balanceOf(address(this));
   uint256 remainingDebt = depositTokenAmount.sub(liquidDeposits);
   if (remainingDebt > 0) {
       for (uint256 i = 0; i < supportedStrategies.length(); i++) {
           address strategy = supportedStrategies.at(i);
           uint256 deployedBalance = getDeployedBalance(strategy);
           if (deployedBalance > remainingDebt) {
               _withdrawFromStrategy(strategy, remainingDebt);
               break;
           } else if (deployedBalance > 0) {
               _withdrawPercentageFromStrategy(strategy, 10000);
               remainingDebt = remainingDebt.sub(deployedBalance);
               if (remainingDebt <= 1) {
                   break;
               }
```

```
            }
        }
    }
    uint256 withdrawAmount = depositToken.balanceOf(address(this)).sub(liquidDeposits);
    depositToken.safeTransfer(msg.sender, withdrawAmount);
    _burn(msg.sender, amount);
    emit Withdraw(msg.sender, depositTokenAmount);
}
```

This issue's severity was lowered because the development team informed us that they will vet which strategies are used on a `YakVaultForSA`.

Recommendation

Use a reentrancy guard for all the public or external non-view functions in YakVault not guarded by the `onlyOwner()` modifier. We recommend this because there may be other reentrancy problems given the code design.

Those are:

- `deposit()` (line 60)
- `depositWithPermit()` (line 64)
- `depositFor()` (line 75)
- `withdraw()` (line 96)

Status

Resolved. All public non-view functions were made non-reentrant. Done for version `51f45e446c3fe95ee98679a68f184d8699b93fee`.

ME-02 YakVaultForSA Denial Of Service

Location:

- `contracts/YakVault.sol`

A rogue strategy may render a useless vault, disallowing deposits and withdrawals made by any user and also disallowing the option for the owner to remove the rogue strategy by reverting the transaction when any function of the strategy is invoked.

This issue's severity was lowered because the development team informed us that they will vet which strategies are used on a `YakVaultForSA`.

Recommendation

While the denial of service cannot be fully avoided without a complete redesign of the `YakVaultForSA`, it can be made transient by allowing the removal of rogue strategies in the `YakVaultForSA.removeStrategy()` function (lines 161-174). In particular, invert the condition in line 169 to first check if the strategy is disabled in the YakRegistry and never try to calculate the deployed balance if so.

The new line should be similar to:

```
yakRegistry.disabledStrategies(strategy) || getDeployedBalance(strategy) == 0
```

Status

Mitigated. Disabled strategies can be removed in the version `51f45e446c3fe95ee98679a68f184d8699b93fee`. With this mitigation, we consider that the issue is minor now.

ME-03 User Unable To Withdraw Funds

Location:

- `contracts/YakVault.sol:98-101`

If a user invokes the `YakVaultForSA.withdraw()` function and the deposit tokens corresponding to the `amount` passed are less than the vault liquid deposits, then the transaction reverts and funds are not withdrawn.

```
uint256 depositTokenAmount = getDepositTokensForShares(amount);
require(depositTokenAmount > 0, "YakVault::withdraw, amount too low");
uint256 liquidDeposits = depositToken.balanceOf(address(this));
uint256 remainingDebt = depositTokenAmount.sub(liquidDeposits);
```

Recommendation

Add extra logic to handle the situation when the liquid deposits are greater than the deposit tokens to be withdrawn.

Status

Resolved. Done for version `51f45e446c3fe95ee98679a68f184d8699b93fee`.

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 0.8.11;`) and keep it up to date. Also, when updating to 0.8 take into account the new semantics for safe math operations.

Status

Acknowledged. The development team informed us that they will upgrade the solidity version in a separate scope.

MI-02 Multiple Overflows in Shares/Tokens Conversions

Location:

- `contracts/YakStrategy.sol:152`
- `contracts/YakStrategy.sol:164`
- `contracts/YakVault.sol:280`
- `contracts/YakVault.sol:294`

The check for zero in those 4 functions has a possible overflow if the multiplication used overflows. For example see the condition in line 294 of YakVault.sol

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

The `mul()` function may overflow if `totalSupply` and `totalDeposits()` are big enough, triggering a permanent denial of service for withdrawals and deposits.

The possible overflow in the multiplication in line 297

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

is not as impactful given that the `amount` is passed by the user on all its usages, including deposits and withdrawals, which can be split in different invocations if necessary. The same analysis can be made on the other 3 locations.

Recommendation

Change all the `if (AAAA.mul(BBBB) == 0)` guards with `if (AAAA == 0 || BBBB == 0)`.

Status

Partially resolved. Resolved in `YakVaultForSA` for version `51f45e446c3fe95ee98679a68f184d8699b93fee`. Will be solved separately for `YakStrategy`.

MI-03 Unregistered Strategy is Enabled

Location:

- `contracts/YakRegistry.sol:47-49`

If the `YakRegistry.isEnabledStrategy()` function receives an address that was not added to the registry it returns `true`.

This issue does not impact the `YakVaultForSA` contract because the `YakRegistry.isActiveStrategy()` function is used in the `YakVaultForSA.addStrategy()` function.

Recommendation

Add check to see if the address passed is in the `strategies` set.

Status

Resolved. The `YakRegistry` contract has been changed to have an `isHaltedRegistry()` function and the `isActiveStrategy()` function has been removed. The uses of `YakRegistry` have been changed to use the new functionality. Done for version `51f45e446c3fe95ee98679a68f184d8699b93fee`.

MI-04 Wrong documentation for YakVaultForSA.setActiveStrategy()

Location:

- `contracts/YakVault.sol:135`

It says that if `address(0)` is passed then instead of changing the active strategy it disables automatic deposits to the default strategy are disabled but this functionality is not implemented properly.

Recommendation

Do not require `address(0)` to be in the `supportedStrategies` set.

Status

Resolved. Done for version `51f45e446c3fe95ee98679a68f184d8699b93fee`.

Enhancements

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

Table

ID	Title	Status
EN-01	Shares/Tokens Conversion Optimization	Not implemented

Details

EN-01 Shares/Tokens Conversion Optimization

Location:

- `contracts/YakVault.sol:279-284,293-298`

The `YakVaultForSA.getDepositTokensForShares()` and `YakVaultForSA.getSharesForDepositTokens()` functions both call twice the `YakVaultForSA.totalDeposits()` function on each invocation. The `YakVaultForSA.totalDeposits()` function iterates over all the registered strategies.

Recommendation

Refactor both functions to call `YakVaultForSA.totalDeposits()` only once on each invocation.

Status

Not implemented. The development team informed us that they will not implement the suggested enhancement.

Other Considerations

Centralization

`YakRegistry`, `YakVaultForSA` and `YakStrategy` are all ownable contracts where the owner has significant powers over the contract, so it must be trusted.

In particular, a `YakVaultForSA` owner has the means to steal the deposit tokens guarded by the vault.

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

Upgradeability

None of the analyzed contracts has provisions to do upgrades.

Iterations, Gas Usage and Denial of Service

The `YakVaultForSA` contract iterates over the supported strategies in several different places (See `YakVault.sol`, lines 103, 125, 268 and 302). This makes most of the operations more expensive as more strategies are supported, so the `YakVaultForSA` owner should keep the number of supported strategies low. And, if the list grows too much, it may even trigger a denial of service because the iteration may exceed the max gas used in a block.

Calculation Errors in Withdrawals

Even when CR-01 and ME-03 are fixed the amount of deposit tokens obtained by a user while doing a `YakVaultForSA.withdraw()` operation may not be correct. The differences may have different sources:

- Misbehaving strategies.
- Rounding errors converting strategy shares and deposit tokens while running `YakVaultForSA.withdraw()`.

Possible loss of funds

If a strategy is removed from the `YakVaultForSA` by the owner using the `removeStrategy()` function, the funds it owned are no longer accounted for in the vault's funds.

Tests

The development team informed us that they have private tests covering the audited contracts outside the git repository.

Changelog

- 2022-01-28 – Initial report based on commit `2a7bc7e0a0ad71fcd60f5d53ab7ed8c350079b93`.
- 2022-02-11 – Fixes checked on commit `51f45e446c3fe95ee98679a68f184d8699b93fee`. Also moved the Tests enhancement proposal to other considerations.

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.