# YieldYak Audit

## Platypus

February 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 `2add9db9263a7bc8e1df8ef184b242d2ed502b1d`.

The audited files are:

- `contracts/strategies/PlatypusVoterProxy.sol`: Contract used by the strategies to interact with the platypus system as a single entity. It only allows registered strategies to interact with the Platypus system.
- `contracts/strategies/PlatypusVoter.sol`: Helper contract used to interact with the Platypus system. The PlatypusVoterProxy uses it for some interactions.
- `contracts/strategies/PlatypusStrategy.sol`: Investment strategy that uses the platypus system.
- `contracts/strategies/PlatypusMasterChefStrategy.sol`: Abstract base contract used to make investment strategies that use the Platypus system. At the time of the audit, `PlatypusStrategy` is its only subcontract.
- `contracts/lib/DSMath.sol`: Library that provides utilities for some arithmetic operations.
- `contracts/lib/ERC20.sol`: ERC20 token with permit implementation.
- `contracts/interfaces/IPlatypusVoterProxy.sol`: Interface definition for the `PlatypusVoterProxy` contract.
- `contracts/interfaces/IPlatypusVoter.sol`: Interface definition for the `PlatypusVoter` contract.
- `contracts/interfaces/IVePTP.sol`: Interface definitions for the contract defined at address `0x5857019c749147EEE22b1Fe63500F237F3c1B692` in the avalanche c-chain. It represents the vePTP token.
- `contracts/interfaces/IPlatypusPool.sol`: Platypus pool interface. Contract documented in https://platypus-finance.gitbook.io/platypus-finance-docs/developers/contracts/pool

- `contracts/interfaces/IMasterPlatypus.sol`: Interface definitions for the contract defined at address `0xB0523f9F473812FB195Ee49BC7d2ab9873a98044` in the avalanche c-chain.
- `contracts/interfaces/IPlatypusAsset.sol`: Platypus asset interface. Contract documented in https://platypus-finance.gitbook.io/platypus-finance-docs/developers/contracts/asset.
- `contracts/interfaces/IWAVAX.sol`: Interface defined to interact with the WAVAX contract.
- `contracts/interfaces/IPair.sol`: Interface defined to interact with decentralized exchange contracts.

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. These files 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:

- `contracts/lib/Ownable.sol`
- `contracts/lib/Context.sol`
- `contracts/lib/EnumerableSet.sol`
- `contracts/lib/SafeERC20.sol`
- `contracts/lib/Address.sol`
- `contracts/lib/IERC20.sol`

The following files were audited in other audits:

- `contracts/lib/DexLibrary.sol`
- `contracts/lib/YakStrategy.sol`
- `contracts/lib/SafeMath.sol`

Also, no tests were reviewed for this audit.

Fixes and improvements were checked in commit `cf8fd8a3088ea877ca6b23f54473a101703011ad`. In this commit also `PlatypusMasterChefStrategy.sol` was merged into `PlatypusStrategy.sol` and the `PlatypusStrategy` contract was made to inherit from `YakStrategyV2` instead of `YakStrategy`, which was reviewed in a different 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, 2 medium issues and several minor issues. Also, several enhancements were proposed.

We checked fixes in commit `cf8fd8a3088ea877ca6b23f54473a101703011ad`. All critical and medium issues were resolved. The minor issues were either resolved or acknowledged. One of the proposed enhancements was implemented.

## Security Issues

| ID | Title | Severity | Status |
|---|---|---|---|
| CR-01 | Race Condition for Rewards | Critical | Resolved |
| ME-01 | Rogue Strategy May Steal Funds of Different Strategy | Medium | Resolved |
| ME-02 | Unguarded Overflows and Underflows | Medium | Resolved |
| MI-01 | Solidity Compiler Version | Minor | Acknowledged |
| MI-02 | Extra-Reward Swap-Pair Not Properly Checked | Minor | Resolved |
| MI-03 | No Symbol Set | Minor | Acknowledged |

# Privileged Roles

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

## ERC20

### 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. This is specified in ERC-20.

## PlatypusVoter

Besides all the roles documented below, it also has the privileged roles documented for the `ERC20` contract.

### Owner

The owner can change the address of the voter proxy.

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

### Dev

The dev can:

- Enable or disable deposits.
- Claim vePTPs.[1]

The dev is passed in the constructor and cannot be changed.

### Voter Proxy

The voter proxy can:

- Claim vePTPs.
- Use the contract balance to do deposits
- Transform AVAX in WAVAX.
- Invoke any function on any contract with any parameters on behalf of the PlatypusVoter.

[1] The dev cannot claim vePTPs anymore in commit `cf8fd8a3088ea877ca6b23f54473a101703011ad`.

## PlatypusVoterProxy

### Dev

The dev can:

- Add approved strategies (but cannot remove them)
- Set booster and staker fees.
- Set fee receivers for booster and staker fees.

The dev is passed in the constructor and cannot be changed.

### Strategy

A strategy can:

- Do deposits
- Withdraw funds
- Do emergency withdraws
- Claim rewards

There are no registered strategies by default.

## PlatypusMasterChefStrategy

It has all the roles defined for the `YakStrategy` contract. No new roles were added. Given that `PlatypusMasterChefStrategy` is an abstract contract, strategies inherited from it may have other roles.

### Owner

Besides all the abilities defined in the parent contracts, the owner can rescue deployed funds. The `setAllowances()` method has been deprecated and if invoked will revert with an error message even if invoked by the owner.

### Dev

Besides all the abilities defined in the parent contracts, the dev can set the extra-reward swap-pair.

### EOA

Besides all the abilities defined in the parent contracts, an EOA can do a reinvestment.

## PlatypusStrategy

It has all the roles defined for the `PlatypusMasterChefStrategy` contract. No new roles were added.

### Owner

Besides all the abilities defined in the parent contracts, the owner can set the Platypus voter proxy.

### Dev

Besides all the abilities defined in the parent contracts, the dev can update the maximum withdraw slippage parameter.

# 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

## CR-01 Race Condition for Rewards

**Location**:

- `contracts/strategies/PlatypusVoterProxy.sol:230`

Strategies that invoke `PlatypusVoterProxy.claimReward()` using the same Platypus PID will take all the rewards associated with that PID, including the ones that should be awarded to another strategy.

### Recommendation

When the dev approves a strategy by calling `PlatypusVoterProxy.aprove()`, it should pass its PID and then the PID should be checked on each call the strategy makes to see that the correct one was used. It is also important not to have 2 different strategies with the same platypus PID, which should be controlled when `PlatypusVoterProxy.aprove()` is executed.

### Status

**Resolved.** In commit `cf8fd8a3088ea877ca6b23f54473a101703011ad`, the `PlatypusVoterProxy.onlyStrategy()` modifier receives the PID and checks if it matches the strategy.

# Medium Severity Issues

## ME-01 Rogue Strategy May Steal Funds of Different Strategy

**Location**:

- `contracts/strategies/PlatypusVoterProxy.sol:166,203`

Nothing in the `PlatypusVoterProxy` code stops a rogue strategy, which was previously approved by the dev, from invoking either `PlatypusVoterProxy.withdraw()` or `PlatypusVoterProxy.emergencyWithdraw()` with parameters corresponding to another strategy and stealing its funds.

The severity of this issue was lowered because the dev[2] is required to approve the rogue strategy in order to be able to do this attack.

[2] See PlatypusVoterProxy roles in the privileged roles section.

### Recommendation

When the dev approves a strategy by calling `PlatypusVoterProxy.aprove()`, it should pass its PID and then the PID should be checked on each call the strategy makes to see that the correct one was used. It is also important not to have 2 different strategies with the same platypus PID, which should be controlled when `PlatypusVoterProxy.aprove()` is executed.

### Status

**Resolved.** In commit `cf8fd8a3088ea877ca6b23f54473a101703011ad`, the `PlatypusVoterProxy.onlyStrategy()` modifier receives the PID and checks if it matches the strategy.

## ME-02 Unguarded Overflows and Underflows

**Location**:

- `contracts/strategies/PlatypusStrategy.sol:141,146,157,159`
- `contracts/lib/DSMath.sol:26,31,54-61,67`
- `contracts/strategies/PlatypusVoterProxy.sol:275`

In several different places there are arithmetic operations that may overflow or underflow. The codebase is in solidity 0.7.3 so the standard operations wrap around when overflowing or underflowing, instead of raising an error.

### Recommendation

Use the `contracts/lib/SafeMath.sol` library for all the arithmetic operations or, even better, upgrade the code to solidity ≥ 0.8 and use the default arithmetic operators, which check for overflows and underflows by default.

### Status

**Resolved**. In commit `cf8fd8a3088ea877ca6b23f54473a101703011ad`, all the arithmetic operations mentioned are using the SafeMath library.

# Minor Severity Issues

## MI-01 Solidity Compiler Version

Most of the 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. A few other files use a non-pinned version, such as `pragma solidity >=0.6.0 <0.8.0;` which also adds risks.

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 Extra-Reward Swap-Pair Not Properly Checked

**Location**:

- `contracts/strategies/PlatypusMasterChefStrategy.sol:103-107`

If the `_extraTokenSwapPair.token0()` is not the reward token, then it is assumed that `_extraTokenSwapPair.token1()` is the reward token. This may lead the strategy to use a wrong swap pair, and instead of getting reward tokens the strategy will get a different ERC20 token.

Hereunder are the lines 103-107 where the issue is located:

```
if (IPair(_extraTokenSwapPair).token0() == address(rewardToken)) {
   extraToken = IPair(_extraTokenSwapPair).token1();
} else {
   extraToken = IPair(_extraTokenSwapPair).token0();
}
```

### Recommendation

Check if `_extraTokenSwapPair.token1()` is the reward token in the else statement on line 106 and revert if necessary.

### Status
**Resolved**. The recommendation was implemented for commit `cf8fd8a3088ea877ca6b23f54473a101703011ad` .

### MI-03 No Symbol Set

**Location**:

- `contracts/strategies/PlatypusStrategy.sol`
- `contracts/strategies/PlatypusMasterChefStrategy.sol`

Neither `PlatypusStrategy` nor its parent, `PlatypusMasterChefStrategy`, set the symbol for the contract, so the symbol defined in the `YakERC20` contract (`"YRC"`) is used, which may lead to confusion. Given that strategies are `ERC20` tokens, a distinguishing symbol must be chosen in order to be properly labeled by several tools, such as MetaMask.

### Recommendation
Pass the symbol as a parameter in the constructors.

### Status
**Acknowledged**. The development team informed us that because the YRT tokens are not meant to be added to Metamask, transferred or traded and they are purely a receipt and helper for the frontend, they don't think resolving this issue is necessary.

# Enhancements

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

## Table

| ID | Title | Status |
|---|---|---|
| EN-01 | Gas Usage Optimization | Implemented |
| EN-02 | Strategy Repudiation | Not implemented |

## Details

### EN-01 Gas Usage Optimization

**Location**:

- `contracts/strategies/PlatypusVoterProxy.sol:20`

If the `SafeProxy.safeExecute()` function fails to run the `platypusVoter.execute()` function all the available gas will be consumed.

This is the offending line:

```
if (!success) assert(false);
```

#### Recommendation

Use `revert` or `require` instead of `assert` to avoid consuming all the remaining available gas.

#### Status

**Implemented**. In commit `cf8fd8a3088ea877ca6b23f54473a101703011ad`, `assert()` was replaced with `revert()`.

### EN-02 Strategy Repudiation

**Location**:

- `contracts/strategies/PlatypusVoterProxy.sol`

The `PlatypusVoterProxy` dev should have the ability to repudiate an approved strategy, disallowing the strategy to further interact with it. This functionality may be used to stop a rogue strategy.

Status
**Not implemented**. The development team informed us that strategy repudiation can be accomplished by migrating the `VoterProxy` if it is required.

# 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

`PlatypusStrategy` is an ownable contract where the owner has significant powers over the contract, so it must be trusted. `PlatypusMasterChefStrategy` is ownable as well, but given that it is an abstract contract it will never be instantiated as is.

`The PlatypusVoter` has an owner, but also a developer with significant responsibility.

The `PlatypusVoterProxy` has an owner responsible to approve strategies to use the `PlatypusVoter`.

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

## Upgradeability

A `PlatypusVoter` owner can change the `PlatypusVoterProxy` to a different contract in order to upgrade the logic, but keeping the `PlatypusVoter` state and assets. The owner of a `PlatypusStrategy` may change the voter proxy used by the strategy to point to the new `PlatypusVoterProxy`.

The rest of the contracts do not have provisions to be upgraded.

## Contract Instantiation

In order to do the security analysis, we assumed that when deploying a `PlatypusStrategy` the `_voterProxy` constructor parameter receives an address of a `PlatypusVoterProxy` contract. We also assumed that when deploying a

`PlatypusVoterProxy` the `_platypusVoter` constructor parameter receives an address of a `PlatypusVoter` contract.

### Extra Fees

As a user of the `PlatypusStrategy`, fees taken by the `PlatypusVoterProxy`, staker fees and booster fees, need to be paid. This is in addition to all the fees that need to be paid as a normal operation for any `YakStrategy`, such as dev fees or reinvest fees.

### Tests

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

## Changelog

- 2022-02-18 – Initial report based on commit `2add9db9263a7bc8e1df8ef184b242d2ed502b1d`.
- 2022-02-21 – Fixes checked on commit `cf8fd8a3088ea877ca6b23f54473a101703011ad`.