PUBLIC

Code Assessment

of the oYfi Smart Contracts

March 7, 2023

Produced for

yearn.finance

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Yearn Team,

Thank you for trusting us to help Yearn with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of oYfi according to Scope to support you in forming an opinion on their security risks.

Yearn implements an incentive mechanism for users to hold the yvTokens. In particular, users can stake these tokens and mint Gauge tokens (ygTokens). With these tokens users can claim Option-Yfi (`oYFI`) which allows them to buy `YFI` tokens on discount.

The most critical subjects covered in our audit are rewards accumulation, the minting and redeeming of Gauge tokens, the calculation of the `YFI` discounted price and, the precision of the calculations and the access control. The security of all aforementioned subjects is high as only low to medium severity issues were uncovered. All the issues have been resolved in the second iteration of the codebase.

The general subjects covered are upgradeability, documentation, testing. The documentation provided to us was limited. The security regarding the rest of subjects is high.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	2
• Code Corrected	2
Low-Severity Findings	2
• Code Corrected	2

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the oYfi repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	13 February 2023	0f2c0c64b396bbf467559d6e23aa8846aa81a10c	Initial Version
2	23 February 2023	59b37862b57cd731359be80d522fda3c279b4e8a	Version with fixes
3	6 March 2023	b5ae35e41dd584e18838202a0146a0eab729c02e	Removed require statement

For the solidity smart contracts, the compiler version `0.8.15` was chosen.

In scope are considered the contracts in the directory `contracts` modified by the commit under review namely:

- `BaseGauge.sol`
- `Gauge.sol`
- `GaugeFactory.sol`
- `OYfi.sol`
- `Options.vy`
- `Registry.sol`

Moreover, `OYfiRewardPool.sol` was audited only concerning its differences to the `RewardPool.sol` which was audited in a different review.

2.1.1 Excluded from scope

Excluded from scope are all the contracts not explicitely mentioned in the scope. More specifically, `OYfiRewardPool.sol` is considered to be functionally correct. All `open-zeppelin` libraries used are also considered to work correctly.Furthermore, all the contracts with which the contracts under scope interact are considered to work as intendend. Finally, attacks by authorized users were not considered in this review as this roles are considered trusted by the system and expected to never act maliciously.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Yearn offers an implementation of the option-YFI (`oYFI`) token. Users earn `oYFI` when they hold Gauge tokens which they can later redeem to get YFI tokens at a discount. Gauge tokens can be minted by depositing yearn vault shares. For example, a user who stakes DAI on Yearn gets vyDAI Vault tokens in return. They can be deposited into Gauge with a mint of yGvyDAI in return. Moreover, they can further boost their rewards by holding `veYFI` Vested Escrow tokens.

2.2.1 Gauge Token

It is a standard ERC20 token and a vault that implements the EIP4626 standard. Users deposit an amount of their shares and mint in return the same amount of the respective Gauge token.

Boosted Balance:

Users can boost their rewards by owning an amount of `veYFI`. Their boosted balance is calculated as follows:

$$min(Gauge.balanceOf(user), \frac{9}{10} * Gauge.totalSupply * \frac{veYFI.balanceOf(user)}{veYFI.totalSupply} + \frac{Gauge.balanceOf(user)}{10})$$

Intuitively, a user will get a higher reward if they hold the same share of all available `veYFI` tokens as their share of all Gauge tokens.

Reward Queueing:

Any user can call `queueNewRewards` function and deposit the reward `oYFI` tokens for distribution in the distribution period. Each distribution period is defined by owner-controlled duration variable. The new period starts if the previous period has ended or if the new queued reward is 1.2 times more than the number of rewards so far distributed. If a user calls `queueNewRewards` with a small amount, then the current reward period is not affected and the amount is queued to be used later together with a bigger amount. When a new period starts, all the queued rewards are used for distribution, however, the reward period does not start automatically. At least one call to `queueNewRewards` need to happen for the new period to start. A change in the period duration (`setDuration`) stops the old period and distributes the remaining rewards in a new period, with the new duration.

Reward Calculation:

Rewards in a period are distributed linearly in time among all the accounts based on their boosted balances. Part of the reward calculation is a penalty calculation. The penalty is the difference between the user's boosted balance, and the maximum possible boosted balance for the user with the given Gauge balance value. The penalty is transferred to the `oYFI` reward pool.

Entry points:

Users can interact with the contract from the following entry points:

- `deposit/mint`: any user can deposit an amount of the underlying token and mint the same amount of gauge tokens for themselves or a specified receiver. Note that since vault shares and gauge tokens are always pegged in a 1-to-1 relation, `mint` and `deposit` implement the same functionality.
- `withdraw/redeem`: any user can redeem their gauge tokens or a pre-specified amount of another user in exchange for the underlying asset. This will lead to burning the respective amount of gauge tokens. Users can optionally choose whether they want to claim their accrued rewards. Again `withdraw` and `redeem` implement the same functionality.
- `getReward`: any user can call this on behalf of any user. It sends the accrued rewards to the specified recipient of the user who earned the rewards. It also updates the reward for the user.
- `kick`: it updates the boosted balance of a batch of users. Any user can call this.

All the aforementioned calls, make a call to `_updateReward` which essentially checkpoints the accrued rewards up to this point.

2.2.2 Gauge Factory

This is a proxy factory, that deploys Gauge proxy contracts, that use some already deployed Gauge contracts as their logic.

2.2.3 oYFI

It is a standard ERC20 token. The token supply can only be minted by its owner. According to specification, the owner in the current architecture is a fully trusted Yearn multisig. Any user can burn any amount of their own `oYFI` tokens or up to a specified amount of another user who has allowed them to do so.

2.2.4 Options contract

Users can burn oYFI tokens and get the same amount `YFI` tokens using the Options contract. For that, they need to call `exercise` function and send some ETH along with the call as a message value. The amount of ETH needed is determined by the `discount * YFI/ETH price` formula. The price of `YFI` is considered to be the maximum between the YFI/ETH price the curve pool price oracle reports and the YFI/ETH price of the Chainlink price oracle. The discount on that price depends on the amount of `YFI` locked on the `veYFI` contract. The greater the amount locked, the smaller the discount. The discount is calculated by the following formula:

$$discount = c/(1 + a * e^{k(x-1)}), where$$
$$c = 1,$$
$$a = 9.9999,$$
$$k = 4.6969$$

The x is the proportion of locked `YFI` tokens, values are between 0 and 1.

The Options contract is not upgradeable. In case an update is needed, the owner of the contract will kill it and transfer its assets to another one.

2.2.5 oYFI reward pool

Users can claim `oYFI` rewards from the oYfiRewardPool contract, based on their historic `veYFI` balance. This contract implements the same functionality as `RewardPool` contract, with disabled relock functionality.

2.2.6 Roles and Trust Model

In the system in scope, the following roles are defined:

- The `oYFI` owner: they can mint `oYFI` tokens. It is assumed that the owner is a multisig controlled by Yearn. It is fully trusted.
- The owner of each Gauge token: they are allowed to update the duration of a period and sweep the contract from tokens that we were accidentally sent there. `oYFI` and `YFI` tokens are protected and cannot be swept.
- The owner of the `Option` contract. They can kill the contract. Fully trusted.

All the aforementioned roles are assumed to never act maliciously or against the interest of the users of the system.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	2

- Missing Sanity Check of Chainlink Oracle Price Code Corrected
- Not Initialized Variables Code Corrected

Low-Severity Findings	2

- Divisions Before Multiplications Code Corrected
- Sweeping Non-ERC20-Compatible Tokens Code Corrected

6.1 Missing Sanity Check of Chainlink Oracle Price

Design Medium Version 1 Code Corrected

`Options` smart contract calculates the required ETH price by querying a YFi/ETH ChainLink oracle and the curve oracle. Apart from the price of the YFi tokens, the oracle returns information about the point in time when its price was updated. However, this information is ignored by the current implementation. The stale prices might be used for estimations.

Code corrected:

A check that the update time of the price oracle complies with the ChainLink heartbeat parameter for the YFI/ETH pool (24 hours or 86400 seconds) was added.

6.2 Not Initialized Variables

Design Medium Version 1 Code Corrected

On multiple occasions, some state variables are used which are never set and there are no functions that can update them. In particular:

- In `Options.exercise`, ETH is sent to `self.payee`. However, this variable is never set. Hence, ETH will be sent to `0x0` address.
- In `Gauge._getReward`, the `recipients` mapping is read. However, this mapping is never written, thus the `recipient[account]` will always be `0x0`. This means, that no other recipient than the owner of the Gauge tokens can receive the rewards.

Code corrected:

- The `Options.payee` is set to the owner in the constructor. In addition, `set_payee` function, restricted to the owner, was added. It can change this field.
- The `Gauge.setRecipient` function was added. It allows users to set the `recipients` mapping.

6.3 Divisions Before Multiplications

Design Low Version 1 Code Corrected

In the implementation in the scope, there are multiple instances where divisions happen before multiplications. Such sequences of operations yield less precise results. In particular, the following expressions can be rearranged:

- In `Options._eth_required`,

```
amount * eth_per_yfi / PRICE_DENOMINATOR * discount / DISCOUNT_NUMERATOR
```

- In `Gauge._boostedBalanceOf`,

```
((_realBalance * BOOSTING_FACTOR) +
    (((totalSupply() * IVotingYFI(VEYFI).balanceOf(_account)) /
        veTotalSupply) *
        (BOOST_DENOMINATOR - BOOSTING_FACTOR))) /
    BOOST_DENOMINATOR,
```

Code partially corrected:

- The `Options._eth_required` does the multiplications first and only then the divisions.
- The `Gauge._boostedBalanceOf` is left unchanged. This numerical imprecision won't affect the functionality of the contract. No "dust" will be accumulated due to this because the penalty is defined in a way, that will sweep the leftover dust.

6.4 Sweeping Non-ERC20-Compatible Tokens

Design Low Version 1 Code Corrected

`Options.sweep` allows any user to transfer any ERC20 compatible tokens owned by the contract to its owner. However, this call will fail for tokens that are not compliant with the ERC20 standard. The most prominent example is the USDT. USDT's `transfer` does not return any value in contrast to the ERC20 standard. This means that `transfer` call will fail. In Solidity, this issue is tackled with the `safeTransfer` call (see Openzeppelin's `safeERC20`).

Code corrected:

The `default_return_value=True` parameter was added in the `Options.sweep` token `transfer` call, that enables `safeTransfer` functionality in Vyper smart contracts.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 EIP-4626 Event Field Names

Informational **Version 1**

Event Deposit and event Withdrawal in IERC4626 are defined with `address indexed caller`. According to the https://eips.ethereum.org/EIPS/eip-4626#events, these fields should be named as `sender`.

7.2 Full YFI Locked Discount Reverts

Informational **Version 1**

In the case of the quite improbable event, when the total supply of Yfi is locked in veYfi, the discount cannot be computed.

```
DISCOUNT_TABLE[total_locked * DISCOUNT_GRANULARITY / total_supply]
```

The `DISCOUNT_TABLE` has 500 elements. However, the max index is 499. This index access during the computation will revert, if `total_locked == total_supply`, because the element with index 500 is not present in the `DISCOUNT_TABLE`.

7.3 Incorrect Documentation

Informational **Version 1**

In `OYfiRewardPool.burn`, the documentation reads as follows:

> @notice Receive YFI into the contract and trigger a token checkpoint

The documentation is incorrect as `OYFI` is transferred instead of `YFI`.

7.4 Missing Indices in Events

Informational **Version 1**

For some events, some arguments are not indexed even if this would make sense. In particular:

- In `Options.Sweep`, the `token` argument could be indexed.
- In `Gauge.BoostedBalanceUpdated`, the `account` argument could be indexed.

7.5 Non-informative Error Message

Informational **Version 1**

`BaseGauge.queueNewRewards` checks whether the `_amount` argument is non 0. Should this check fail, the non-informative `==0` message will be returned. Note that Solidity 0.8 allows for error values instead of just strings to be returned upon check failure.

7.6 Redundant Function Modifiers

Informational **Version 1**

Multiple functions are defined using a public modifier, while they are not called within the contract. These functions could be set as external instead:

- `Gauge.convertToShares`
- `Gauge.convertToAssets`
- `Gauge.maxDeposit`
- `Gauge.previewDeposit`
- `Gauge.maxMint`
- `Gauge.previewMint`
- `Gauge.kick`

Solidity compiler needs to perform extra routines for public functions, which can result in higher gas usage.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Gauge Assumed Decimals

Note Version 1

The `Gauge` contract uses default 18 decimals. However, the asset can have a different number of decimals. While the Yearn vault tokens have 18 decimals, this might not be true for any asset that might be used in Gauge. If an asset with a different number of decimals is introduced, the respective Gauge will misbehave.

8.2 Manipulation of Curve Oracle

Note Version 1

`Options` calculates the price of YFI/ETH by querying the price oracle of the respective Curve-pool. Curve uses a time weighted price oracle or (TWAP-oracle). TWAP oracles have been shown to be manipulatable to an extent. Users should be aware that the system in scope does not perform any further sanity checks on the correctness of the reported price.