PUBLIC

Code Assessment

of the Partner Tracker Smart Contracts

18 January, 2022

Produced for

yearn.finance

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Yearn team,

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

Yearn Finance implements a partner tracker that tracks the vault deposits done over an affiliate partner. The tracked amount is simply the sum of all funds deposited via the partner into a specific vault.

We did not uncover any security related issues. Minor issues like unused imports or constants were found. Also, for a small code base we recommend to follow best practices and comment as well as document the code accordingly. All issues found were fixed accordingly.

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	0
Low-Severity Findings	5
• Code Corrected	5

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 Partner Tracker 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	12 January 2022	3c8cc0e68e44e168590846f2d676cb1a0c5bc7cc	Initial Version
2	18 January 2022	6937685f3748d6017c0f7498c32c6491f330609e	Second Version

For the solidity smart contracts, the compiler version `0.6.12` was implicitly chosen in the brownie configuration file.

2.1.1 Excluded from scope

The called contracts like the vault and token contracts.

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 Finance implements a partner tracking program in a smart contract. This contract allows end users to deposit funds into a vault and specify a partner id, which is used to track deposits. The contract has only the `deposit` functionality. There are two functions:

- `deposit(vault,partnerId)`: deposits into a vault the whole balance of `msg.sender` in the respective token.
- `deposit(vault,partnerId,amount)`: deposits into a vault the `amount` specified by the user.

The contract does not have specific roles, so any user can call the above functions. The input parameters of the functions are not validated, so the mapping `referredBalance` can be populated with arbitrary addresses and the respective events are emitted. However, we assume the legitimate vaults of Yearn behave correctly and other applications reading the mapping or the events, filter out the incorrect ones.

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	0
Low-Severity Findings	5

- Event Parameters Not Indexed Code Corrected
- Missing Code Comments and Function Descriptions Code Corrected
- Missing Return Values Code Corrected
- Unused Constant registry Code Corrected
- Unused Imports Code Corrected

6.1 Event Parameters Not Indexed

Design Low Version 1 Code Corrected

The event `ReferredBalanceIncreased` emits the address of `partnerId`, `vault` and `depositer`. All three information might be relevant to later query specific deposits. Hence, it might be useful to index these parameters.

Code corrected:

The updated event now indexes the parameters: `partnerId`, `vault` and `depositer`.

6.2 Missing Code Comments and Function Descriptions

Design Low Version 1 Code Corrected

Even though the code base is simple and well structured, code comments as well as function description with parameter descriptions are part of good coding practice to help understanding the code. The current implementation lacks documentation and code comments.

Code corrected:

The updated code contains specifications that describe the functions and their parameters.

6.3 Missing Return Values

Design | Low | Version 1 | Code Corrected

Both external functions `deposit` declare a return value of type `uint256`, but the return statement is missing.

Code corrected:

The internal function `_internalDeposit` is modified to return `receivedShares`, which is then returned by both external functions `deposit`.

6.4 Unused Constant `registry`

Design | Low | Version 1 | Code Corrected

The constant `registry` is defined but not used in the current code base.

Code corrected:

The unused constant has been removed from the updated code.

6.5 Unused Imports

Design | Low | Version 1 | Code Corrected

The Math and Address libraries and the registry interface are imported but not used.

```
import "@openzeppelin/contracts/math/Math.sol";
import "@openzeppelin/contracts/utils/Address.sol";
import "../interfaces/IYearnRegistry.sol";
```

Code corrected:

The unused imports have been removed from the updated code.

7 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.

7.1 Contract Tracks Also Malicious Vaults and Tokens

Note Version 1

The input arguments for the vault to be deposited in and the token are provided by the user. Both can be malicious contracts. We could not see a way to exploit the contract, but the mapping will record everything including the invalid/malicious vaults. Hence, when reading the mapping the correct vaults needs to be carefully selected and invalid records neglected.

7.2 Outdated Compiler Version

Note Version 1

The compiler version is outdated (https://swcregistry.io/docs/SWC-102) and implicitly fixed in the brownie config file to `version: 0.6.12`. The contract has a floating pragma for the compiler version, although practically there is no newer version without breaking changes (https://swcregistry.io/docs/SWC-103). This version has the following known bugs: https://docs.soliditylang.org/en/v0.6.12/bugs.html

This is just a note as we do not see any severe issue using this compiler with the current code.