YEARN VAULT V2 SMART CONTRACT AUDIT

(Solidity part)

December 3, 2020

MixBytes()

TABLE OF CONTENTS

INTRODUCTION TO THE AUDIT 2
General provisions 2
Scope of audit 2
SECURITY ASSESSMENT PRINCIPLES 3
Classification of issues 3
Security assessment methodology 3
DETECTED ISSUES 4
Critical 4
Major 4
Warning 4
1. No validation of address parameter value in contract constructor 4
2. No validation of the address parameter value in function before using this parameter in access modifiers 5
3. Ability to change the value of a private variable from another contract 6
4. Function calculation result is not processed 6
5. Safe math is not used 7
6. Function description differs from implementation 7
Comments 8
1. Duplicate code 8
2. Variable declared but not used 9
3. No event registration when changing the parameters of the contract 9
4. Exact block timestamps usage 9
CONCLUSION AND RESULTS 10
ABOUT MIXBYTES 10
DISCLAIMER 10

01 INTRODUCTION TO THE AUDIT

General Provisions

Yearn Finance is a decentralized investment aggregator that leverages composability and uses automated strategies to earn high yield on crypto assets.
The audited contract is a part of a new second version of Yearn vaults. Yearn vaults represent a user funds manager in Yearn ecosystem.
Smart contract itself is a base contract for strategies. It defines strategy interface and provides common functionality and restrictions for them.
The contract is designed to be overridden by particular strategy and allows to implement any custom logic and at same time one may not worry about interface compatibility.

Scope of audit

The scope of the audit includes the following smart contract at:
BaseStrategy.sol

The audited commit identifiers:

- `54db126821c4d7aaaf5839be935cecb9b1bf088b`
- `cff924f1894cca1820a588b14d341c4fa4f384c0`

02 SECURITY ASSESSMENT PRINCIPLES

Classification of Issues

- CRITICAL: Bugs leading to Ether or token theft, fund access locking or any other loss of Ether/tokens to be transferred to any party (for example, dividends).

- MAJOR: Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.

- WARNINGS: Bugs that can break the intended contract logic or expose it to DoS attacks.

- COMMENTS: Other issues and recommendations reported to/ acknowledged by the team.

Security Assessment Methodology

Two auditors independently verified the code.

Stages of the audit were as follows:

- "Blind" manual check of the code and its model
- "Guided" manual code review
- Checking the code compliance with the customer requirements
- Discussion of independent audit results
- Report preparation

03 DETECTED ISSUES

CRITICAL

Not found

MAJOR

Not found

WARNINGS

1. No validation of address parameter value in contract constructor

Description

There is no more functionality to initialize the `vault` variable in the contract. If the value of the variable passed to **BaseStrategy.sol#L162** is not correct, then the contract will not work.

Recommendation

We suggest adding a parameter check.

Status

Acknowledged

2. No validation of the address parameter value in function before using this parameter in access modifiers.

Description

At this function: **BaseStrategy.sol#L171**

```
function setStrategist(address _strategist) external {
  require(msg.sender == strategist || msg.sender == governance(),
"!authorized");
  strategist = _strategist;
}
```

There is a possibility that the `strategist` variable and the `governance()` function value will simultaneously contain zero or an invalid address value. Then the work of the smart contract will be blocked.

Recommendation

We suggest adding a parameter check.

Status

Acknowledged

3. Ability to change the value of a private variable from another contract

Description

The private variable `reserve` has a lot to do with the logic behind this smart contract. The value of this variable is changed in this contract. If you change the value of this variable from another contract, then the basic logic of the smart contract may be violated, up to the complete stop of the smart contract.
Considering that the setter for this variable `setReserve()` defined **BaseStrategy.sol#L156** is an internal function and there is no event logging when the value of this variable is changed.
In addition, according to the rules of clean architecture and the principles of software development SOLID (single responsibility, open-closed, Liskov substitution, interface segregation и dependency inversion), each module should be responsible for only one functionality. It is not correct when, in addition to the BaseStrategy contract, the logic of the `harvest()` and `setEmergencyExit()` functions will be changed by another contract created for other purposes.

Recommendation

We recommend removing this functionality.

Status

Fixed at **cff924f1**

4. Function calculation result is not processed

Description

Below there will be a case when the code does not process the result of calling approve.
BaseStrategy.sol#L164

approve method may return false.

Recommendation

We recommend handling return value

Status

Acknowledged

5. Safe math is not used

Description

At this line: **BaseStrategy.sol#L338**

```
return (profitFactor * callCost < credit.add(profit));
```

Due to the fact that safe math is not used, overflow and incorrect logic of the `harvestTrigger()` function may occur.

Recommendation

We suggest using SafeMath

Status

Fixed at **PR-107**

6. Function description differs from implementation.

Description

The function **BaseStrategy.sol#L311** says that

> this call and `tendTrigger` should never return `true` at the same time.

When returning the result, you need to add a check for the value of the `tendTrigger()` function

Recommendation

We recommend explicitly checking the required invariant, in the case if we want to allow overriding these functions you can use an approach like here: https://gist.github.com/algys/edf4f05c01ffd023190d3acca6982259

Status

Acknowledged

COMMENTS

1. Duplicate code

Description

Duplicate code, i.e. using the same code structures in several places. Combining these structures will improve your code. The use of duplicate code structures impairs the perception of the program logic and can easily lead to errors in subsequent code edits. Duplicate code violates SOLID (single responsibility, open-closed, Liskov substitution, interface segregation и dependency inversion) software development principles.

BaseStrategy.sol
Line 170, 175, 180, 185, 190, 383
Instead of this code:
`require(msg.sender == strategist || msg.sender == governance(), "!authorized");`
we recommend making the access modifier

```
modifier onlyStrategistOrGovernance() {
    require(msg.sender == strategist || msg.sender == governance(),
"!authorized");
    _;
}
```

Line 280, 329
Instead of this code:

```
if (keeper != address(0)) {
            require(msg.sender == keeper || msg.sender == strategist ||
msg.sender == governance(), "!authorized");
}
```

Recommendation

We recommend moving the code into a separate function

Status

Fixed at PR-107

2. Variable declared but not used

Description

At this line: **BaseStrategy.sol#L273**
The `callCost` variable is not used

Recommendation

We recommend removing unused variables

Status

Acknowledged

3. No event registration when changing the parameters of the contract

Description

In file: **BaseStrategy.sol**
Lines 156, 169, 174, 179, 184, 189, 382
There are functions that change contract fields without any event emission.

Recommendation

We recommend adding logging of actions so that you know who changed what.

Status

Fixed at PR-107

4. Exact block timestamps usage.

Description

At this line: **BaseStrategy.sol#L320**
Dependency on particular block exact timestamps can eventually break the business logic because the timestamps not only may vary within the single replication time interval, but they can also be maliciously manipulated. As the result, the output of `harvestTrigger()` will vary every time it gets called.

Recommendation

We recommend to refactor the business logic to be reliant not on a particular block timestamp, but on the replication interval time range.

Status

Acknowledged

04 CONCLUSION AND RESULTS

Findings list

Level	Amount
CRITICAL	0
MAJOR	0
WARNING	6
COMMENT	4

Final commit identifier with all fixes: `99dcc2a8ce495ac6c2ff08e633e5b475a3088255`

About MixBytes

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build open-source solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://t.me/MixBytes

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Yearn Finance. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.