Yearn Finance V2

Score: 93%

Overview

This is a Yearn Finance Process Quality Review completed on July 28th 2021. It was performed using the Process Review process (version 0.7.3) and is documented here. The review was performed by Nic of DeFiSafety. Check out our Telegram.

The final score of the review is 93%, an excellent pass. The breakdown of the scoring is in Scoring Appendix. For our purposes, a pass is **70%.**

Summary of the Process

Very simply, the review looks for the following declarations from the developer's site. With these declarations, it is reasonable to trust the smart contracts.

- **Here are my smart contracts on the blockchain**
- **Here is the documentation that explains what my smart contracts do**
- **Here are the tests I ran to verify my smart contract**
- **Here are the audit(s) performed on my code by third party experts**
- **Here are the admin controls and strategies**

Disclaimer

This report is for informational purposes only and does not constitute investment advice of any kind, nor does it constitute an offer to provide investment advisory or other services. Nothing in this report shall be considered a solicitation or offer to buy or sell any security, token, future, option or other financial instrument or to offer or provide any investment advice or service to any person in any jurisdiction. Nothing contained in this report constitutes investment advice or offers any opinion with respect to the suitability of any security, and the views expressed in this report should not be taken as advice to buy, sell or hold any security. The information in this report should not be relied upon for the purpose of investing. In

preparing the information contained in this report, we have not taken into account the investment needs, objectives and financial circumstances of any particular investor. This information has no regard to the specific investment objectives, financial situation and particular needs of any specific recipient of this information and investments discussed may not be suitable for all investors.

Any views expressed in this report by us were prepared based upon the information available to us at the time such views were written. The views expressed within this report are limited to DeFiSafety and the author and do not reflect those of any additional or third party and are strictly based upon DeFiSafety, its authors, interpretations and evaluation of relevant data. Changed or additional information could cause such views to change. All information is subject to possible correction. Information may quickly become unreliable for various reasons, including changes in market conditions or economic circumstances.

Chain

This section indicates the blockchain used by this protocol.

Chain: Ethereum

Guidance:

Ethereum
Binance Smart Chain
Polygon
Avalanche

Code and Team

This section looks at the code deployed on the Mainnet that gets reviewed and its corresponding software repository. The document explaining these questions is here. This review will answer the following questions:

1) Are the executing code addresses readily available? (%)
2) Is the code actively being used? (%)
3) Is there a public software repository? (Y/N)
4) Is there a development history visible? (%)
5) Is the team public (not anonymous)? (Y/N)

1) Are the executing code addresses readily available? (%)

Answer: 100%

They are available at website https://docs.yearn.finance/developers/deployed-contracts-registry, as indicated in the Appendix.

Note: A lot more smart contract addresses can be found here.

Guidance:

100%	Clearly labelled and on website, docs or repo, quick to find
70%	Clearly labelled and on website, docs or repo but takes a bit of looking
40%	Addresses in mainnet.json, in discord or sub graph, etc
20%	Address found but labeling not clear or easy to find
0%	Executing addresses could not be found

2) Is the code actively being used? (%)

Answer: 100%

Activity is over 10 transactions a day on contract *yVault.sol*, as indicated in the Appendix.

Guidance:

100%	More than 10 transactions a day
70%	More than 10 transactions a week
40%	More than 10 transactions a month
10%	Less than 10 transactions a month
0%	No activity

3) Is there a public software repository? (Y/N)

> **Answer:** Yes

GitHub: https://github.com/yearn/yearn-vaults.

Is there a public software repository with the code at a minimum, but also normally test and scripts. Even if the repository was created just to hold the files and has just 1 transaction, it gets a **"Yes"**. For teams with private repositories, this answer is **"No"**.

4) Is there a development history visible? (%)

> **Answer:** 100%

With 320 commits and 5 branches, this is a very healthy software repository.

This metric checks if the software repository demonstrates a strong steady history. This is normally demonstrated by commits, branches and releases in a software repository. A healthy history demonstrates a history of more than a month (at a minimum).

Guidance:

100%	Any one of 100+ commits, 10+branches
70%	Any one of 70+ commits, 7+branches

50%	Any one of 50+ commits, 5+branches
30%	Any one of 30+ commits, 3+branches
0%	Less than 2 branches or less than 30 commits

5) Is the team public (not anonymous)? (Y/N)

> **Answer:** Yes

Location: https://www.linkedin.com/company/iearn-finance/about/.

For a **"Yes"** in this question, the real names of some team members must be public on the website or other documentation (LinkedIn, etc). If the team is anonymous, then this question is a **"No"**.

Documentation

This section looks at the software documentation. The document explaining these questions is here.

Required questions are;

6) Is there a whitepaper? (Y/N)
7) Are the basic software functions documented? (Y/N)
8) Does the software function documentation fully (100%) cover the deployed contracts? (%)
9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)
10) Is it possible to trace from software documentation to the implementation in code (%)

6) Is there a whitepaper? (Y/N)

Answer: Yes

Location: https://docs.yearn.finance/.

7) Are the basic software functions documented? (Y/N)

Answer: Yes

Yearn have their basic software functions documented in their yVault documentation and developer documentation.

8) Does the software function documentation fully (100%) cover the deployed contracts? (%)

Answer: 100%

All of Yearn's contracts and functions are documented in the Naming Conventions section of their documentation, as well as the yVaults section and the API section. In addition, Yearn has robust software documentation in each of the README.md of their various GitHub repositories, most notably yearn-vaults. Lastly, Yearn has a developer section in their website that goes over each software function of each major contract.

Guidance:

100%	All contracts and functions documented
80%	Only the major functions documented
79-1%	Estimate of the level of software documentation
0%	No software documentation

9) Are there sufficiently detailed comments for all

functions within the deployed contract code (%)

> **Answer: 8**0%

The CtC seems to understate how much commenting there is on the Vyper code. For this reason the score was increase to 80%.

Code examples are in the Appendix. As per the SLOC, there is 50% commenting to code (CtC).

The Comments to Code (CtC) ratio is the primary metric for this score.

Note: The Vyper files were invisible to our calculation tools, so therefore we upgraded the score to 50% due to missing files. In addition, we only factored the core contract repositories in our calculations, such as the yearn-vaults, yearn-protocol, and brownie-wrapper-mix repositories. Files such as interface, mocks, and any third-party files were not factored into our calculations.

Guidance:

100%	CtC > 100	Useful comments consistently on all code
90-70%	CtC > 70	Useful comment on most code
60-20%	CtC > 20	Some useful commenting
0%	CtC < 20	No useful commenting

How to improve this score

This score can improve by adding comments to the deployed code such that it comprehensively covers the code. For guidance, refer to the SecurEth Software Requirements.

10) Is it possible to trace from software documentation to the implementation in code (%)

> **Answer:** 100%

There is clear and explicit traceability between the Yearn devdocs and the software functions' implementations in the source code.

Guidance:

100% Clear explicit traceability between code and documentation at a requirement level for all code
60% Clear association between code and documents via non explicit traceability
40% Documentation lists all the functions and describes their functions
0% No connection between documentation and code

Testing

This section looks at the software testing available. It is explained in this document. This section answers the following questions;

11) Full test suite (Covers all the deployed code) (%)
12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)
13) Scripts and instructions to run the tests (Y/N)
14) Report of the results (%)
15) Formal Verification test done (%)
16) Stress Testing environment (%)

11) Is there a Full test suite? (%)

> **Answer:** 80%

Code examples are in the Appendix. As per the SLOC, there is 60% testing to code (TtC).

This score is guided by the Test to Code ratio (TtC). Generally a good test to code ratio is over 100%. However the reviewers best judgement is the final deciding

factor.

Note: Some Python files were missed/miscalculated by our calculation tool so the score is upgraded to 80%.

Guidance:

100% TtC > 120% Both unit and system test visible
80% TtC > 80% Both unit and system test visible
40% TtC < 80% Some tests visible
0% No tests obvious

How to improve this score:

This score can improved by adding tests to fully cover the code. Document what is covered by traceability or test results in the software repository.

12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)

> **Answer:** 100%

Detailed code coverage can be found in their v2 vault repository commits.

Guidance:

100% Documented full coverage
99-51% Value of test coverage from documented results
50% No indication of code coverage but clearly there is a reasonably complete set of tests
30% Some tests evident but not complete
0% No test for coverage seen

13) Scripts and instructions to run the tests (Y/N)

Answer: Yes

Scrips/Instructions location: You can find scripts here and instructions to run tests in the README.md of their various repositories.

14) Report of the results (%)

Answer: 100%

Detailed test report was found at https://github.com/yearn/yearn-vaults/commit/ca36f0113e32abb78e44f61fa95c0f1258daeffd. There is an additional report in the actions of the year-vaults repository here.

Guidance:

100%	Detailed test report as described below
70%	GitHub code coverage report visible
0%	No test report evident

15) Formal Verification test done (%)

Answer: 0%

No evidence of a Formal Verification test was found in the Yearn documentation or in web searches.

16) Stress Testing environment (%)

Answer: 50%

There is evidence of Yearn's test-net usage in the "v2 yVault Improvements" section of their documentation. However, no addresses are publicly available, and therefore this metric's score is reduced to 50%.

Security

This section looks at the 3rd party software audits done. It is explained in this document. This section answers the following questions;

17) Did 3rd Party audits take place? (%)
18) Is the bounty value acceptably high?

17) Did 3rd Party audits take place? (%)

> **Answer:** 100%

Yearn has had two MixBytes audits and one Martinet Lee audit pre-deployment, as well as one Trail of Bits audit post-deployment. Full list of available audit reports can be found here.

Fix recommendations were successfully implemented by the Yearn team.

Guidance:

100% Multiple Audits performed before deployment and results public and implemented or not required
90% Single audit performed before deployment and results public and implemented or not required
70% Audit(s) performed after deployment and no changes required. Audit report is public

50% Audit(s) performed after deployment and changes needed but not

implemented
20% No audit performed
0% Audit Performed after deployment, existence is public, report is not public and
no improvements deployed OR smart contract address' not found, question

Deduct 25% if code is in a private repo and no note from auditors that audit is applicable to deployed code

18) Is the bounty value acceptably high (%)

> **Answer:** 70%

Yearn Finance's Bug Bounty Program with Immunefi is active and rewards participating users with up to 200k for the most critical of finds.

Guidance:

100% Bounty is 10% TVL or at least $1M AND active program (see below)
90% Bounty is 5% TVL or at least 500k AND active program
80% Bounty is 5% TVL or at least 500k
70% Bounty is 100k or over AND active program
60% Bounty is 100k or over
50% Bounty is 50k or over AND active program
40% Bounty is 50k or over
20% Bug bounty program bounty is less than 50k
0% No bug bounty program offered

An active program means that a third party (such as Immunefi) is actively driving hackers to the site. An inactive program would be static mentions on the docs.

Access Controls

This section covers the documentation of special access controls for a DeFi protocol. The admin access controls are the contracts that allow updating contracts or coefficients in the protocol. Since these contracts can allow the protocol admins to "change the rules", complete disclosure of capabilities is vital for user's transparency. It is explained in this document. The questions this section asks are as follow;

19) Can a user clearly and quickly find the status of the admin controls?
20) Is the information clear and complete?
21) Is the information in non-technical terms that pertain to the investments?
22) Is there Pause Control documentation including records of tests?

19) Can a user clearly and quickly find the status of the access controls (%)

Answer: 100%

Yearn's access controls can easily be found in the "Governance" section of their documentation.

Guidance:

100%	Clearly labelled and on website, docs or repo, quick to find
70%	Clearly labelled and on website, docs or repo but takes a bit of looking
40%	Access control docs in multiple places and not well labelled
20%	Access control docs in multiple places and not labelled
0%	Admin Control information could not be found

20) Is the information clear and complete (%)

Answer: 90%

a) All contracts are clearly labelled as upgradeable in the first two sentences.

b) MultiSig and defined roles are detailed in "Governance and Operations".

c) Capabilities for contract change is detailed in "Governance and Operations".

Guidance:

All the contracts are immutable -- 100% OR

a) All contracts are clearly labelled as upgradeable (or not) -- 30% AND
b) The type of ownership is clearly indicated (OnlyOwner / MultiSig / Defined Roles) -- 30% AND
c) The capabilities for change in the contracts are described -- 30%

21) Is the information in non-technical terms that pertain to the investments (%)

Answer: 90%

All Yearn governance information is detailed in user-friendly language.

Guidance:

100% All the contracts are immutable
90% Description relates to investments safety and updates in clear, complete non-software l
language
30% Description all in software specific language
0% No admin control information could not be found

22) Is there Pause Control documentation including records of tests (%)

Answer: 80%

yGuard is explained clearly in the "Governance and Operations" section of the Yearn documentation.

Guidance:

100% All the contracts are immutable or no pause control needed and this is explained OR
100% Pause control(s) are clearly documented and there is records of at least one test
within 3 months

80% Pause control(s) explained clearly but no evidence of regular tests
40% Pause controls mentioned with no detail on capability or tests
0% Pause control not documented or explained

How to improve this score:

Create a document that covers the items described above in plain language that investors can understand. An example is enclosed.

Appendices

Author Details

The author of this review is Rex of DeFi Safety.

Email : rex@defisafety.com Twitter : @defisafety

I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value. Aerospace knows how to make reliable code using quality processes.

I was coaxed to go to EthDenver 2018 and there I started SecuEth.org with Bryant and Roman. We created guidelines on good processes for blockchain code development. We got EthFoundation funding to assist in their development.

Process Quality Reviews are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

DeFiSafety is my full time gig and we are working on funding vehicles for a permanent staff.

Scoring Appendix

	Total	Yearn	
PQ Audit Scoring Matrix (v0.7)	**Points**	**Answer**	**Points**
Total	**260**		240.5
Code and Team			**93%**
1) Are the executing code addresses readily available? (%)	**20**	100%	20
2) Is the code actively being used? (%)	**5**	100%	5
3) Is there a public software repository? (Y/N)	**5**	Y	5
4) Is there a development history visible? (%)	**5**	100%	5
5) Is the team public (not anonymous)? (Y/N)	**15**	Y	15
Code Documentation			
6) Is there a whitepaper? (Y/N)	**5**	Y	5
7) Are the basic software functions documented? (Y/N)	**10**	Y	10
8) Does the software function documentation fully (100%) cover the deployed contracts? (%)	**15**	100%	15
9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)	**5**	80%	4
10) Is it possible to trace from software documentation to the implementation in code (%)	**10**	100%	10
Testing			
11) Full test suite (Covers all the deployed code) (%)	**20**	80%	16
12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)	**5**	100%	5
13) Scripts and instructions to run the tests? (Y/N)	**5**	Y	5
14) Report of the results (%)	**10**	100%	10
15) Formal Verification test done (%)	**5**	0%	0
16) Stress Testing environment (%)	**5**	50%	2.5
Security			
17) Did 3rd Party audits take place? (%)	**70**	100%	70
18) Is the bug bounty acceptable high? (%)	**10**	70%	7
Access Controls			
19) Can a user clearly and quickly find the status of the admin controls	**5**	100%	5
20) Is the information clear and complete	**10**	90%	9
21) Is the information in non-technical terms	**10**	90%	9
22) Is there Pause Control documentation including records of tests	**10**	80%	8
Section Scoring			
Code and Team	50	100%	
Documentation	45	98%	
Testing	50	77%	
Security	80	96%	
Access Controls	35	89%	

Executing Code Appendix

Token Contracts

YFI

The Yearn ecosystem is controlled by YFI token holders who submit and vote on proposals that govern the ecosystem.

Token	Address
YFI	0x0bc529c00c6401aef6d220be8c6ea1667f6ad93e

WOOFY

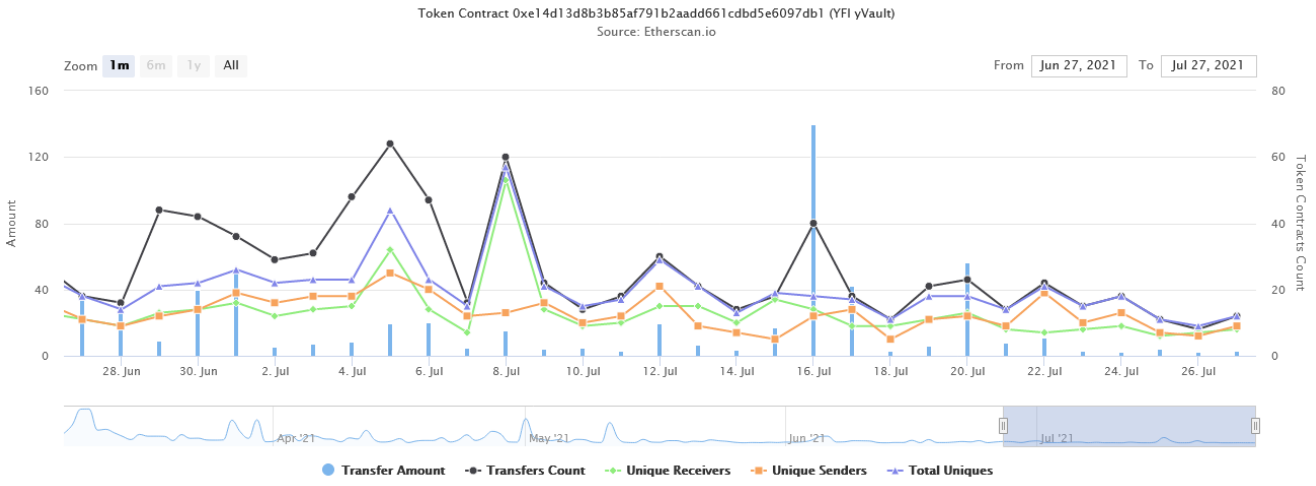
YFI can be transformed into WOOFY through Woofy Finance. It is effectively the same thing as YFI with a smaller denomination. The rate is 1 YFI = 1,000,000 WOOFY.

Token	Address
WOOFY	0xd0660cd418a64a1d44e9214ad8e459324d8157f1

V2 Registry

The Vault Registry is the single source of truth for active Yearn vaults. The registry allows users to query for active Yearn vaults and vault metadata. For info on yield tokens, vault contracts, strategy contracts, and vault statuses or info, check the v2.registry.ychad.eth contract.

Code Used Appendix



Example Code Appendix

```
1  contract yVault is ERC20, ERC20Detailed {
2      using SafeERC20 for IERC20;
```

```
3       using Address for address;
4       using SafeMath for uint256;
5
6       IERC20 public token;
7
8       uint256 public min = 9500;
9       uint256 public constant max = 10000;
10
11      address public governance;
12      address public controller;
13
14      constructor(address _token, address _controller)
15          public
16          ERC20Detailed(
17              string(abi.encodePacked("yearn ", ERC20Detailed(_token).name(
18              string(abi.encodePacked("y", ERC20Detailed(_token).symbol()))
19              ERC20Detailed(_token).decimals()
20          )
21      {
22          token = IERC20(_token);
23          governance = msg.sender;
24          controller = _controller;
25      }
26
27      function balance() public view returns (uint256) {
28          return token.balanceOf(address(this)).add(IController(controller)
29      }
30
31      function setMin(uint256 _min) external {
32          require(msg.sender == governance, "!governance");
33          min = _min;
34      }
35
36      function setGovernance(address _governance) public {
37          require(msg.sender == governance, "!governance");
38          governance = _governance;
39      }
40
41      function setController(address _controller) public {
42          require(msg.sender == governance, "!governance");
43          controller = _controller;
44      }
45
46      // Custom logic in here for how much the vault allows to be borrowed
47      // Sets minimum required on-hand to keep small withdrawals cheap
48      function available() public view returns (uint256) {
49          return token.balanceOf(address(this)).mul(min).div(max);
50      }
51
52      function earn() public {
53          uint256 _bal = available();
54          token.safeTransfer(controller, _bal);
```

```
55          IController(controller).earn(address(token), _bal);
56      }
57
58      function depositAll() external {
59          deposit(token.balanceOf(msg.sender));
60      }
61
62      function deposit(uint256 _amount) public {
63          uint256 _pool = balance();
64          uint256 _before = token.balanceOf(address(this));
65          token.safeTransferFrom(msg.sender, address(this), _amount);
66          uint256 _after = token.balanceOf(address(this));
67          _amount = _after.sub(_before); // Additional check for deflationar
68          uint256 shares = 0;
69          if (totalSupply() == 0) {
70              shares = _amount;
71          } else {
72              shares = (_amount.mul(totalSupply())).div(_pool);
73          }
74          _mint(msg.sender, shares);
75      }
76
77      function withdrawAll() external {
78          withdraw(balanceOf(msg.sender));
79      }
80
81      // Used to swap any borrowed reserve over the debt limit to liquidate
82      function harvest(address reserve, uint256 amount) external {
83          require(msg.sender == controller, "!controller");
84          require(reserve != address(token), "token");
85          IERC20(reserve).safeTransfer(controller, amount);
86      }
87
88      // No rebalance implementation for lower fees and faster swaps
89      function withdraw(uint256 _shares) public {
90          uint256 r = (balance().mul(_shares)).div(totalSupply());
91          _burn(msg.sender, _shares);
92
93          // Check balance
94          uint256 b = token.balanceOf(address(this));
95          if (b < r) {
96              uint256 _withdraw = r.sub(b);
97              IController(controller).withdraw(address(token), _withdraw);
98              uint256 _after = token.balanceOf(address(this));
99              uint256 _diff = _after.sub(b);
100             if (_diff < _withdraw) {
101                 r = b.add(_diff);
102             }
103         }
104
105         token.safeTransfer(msg.sender, r);
106     }
```

```
107
108       function getPricePerFullShare() public view returns (uint256) {
109           return balance().mul(1e18).div(totalSupply());
110       }
111   }
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complexity
Solidity	10	3093	479	729	1885	290

Comments to Code 729/1885 = 39%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complexity
JavaScript	16	1590	295	164	1131	29

Tests to Code 1139/1885 = 60%