Yearn.Finance Security Review

Quantstamp Labs
July 24, 2020

Share this article on:

Introduction

Quantstamp completed its informal code review of Yearn Finance. Yearn Finance provides yield-maximizing opportunities for liquidity providers, and is intended to be governed in a decentralized manner. Due to the large number of contracts involved in the yEarn system, we limited our review to the most prominent contracts—those that hold funds or can distribute funds. We performed this review as a service to the community. Findings are divided by contract below.

Acknowledgements: We want to especially thank Devops199fan, Klim K, Angel, Michael Egorov, Samczsun, vasiliy, Will Price, Cooper Turley, Damir Bandalo and others in the community for reaching out, walking through the code, and gathering needed documentation, as well as Andre Cronje for creating this grand experiment in decentralization. We are looking forward to seeing how this develops.

We hope this is useful documentation to community members that seek to understand and improve yearn. We understand yearn is a work in progress undergoing rapid iteration, and so is this review, we hope it is helpful.

Overview

The code base for this review was relatively challenging due to the following reasons:

- The files contain very few code comments and there was no technical specification available other than posts on Medium or other websites.
- The Solidity files were flattened by default, and each of them therefore has a copy of common contracts such as SafeMath, ERC20, etc. However, it is not trivial to compute a diff of all of these files with the standard implementations of SafeMath, ERC20, etc., because the files may use different versions of these standard contracts and in some places code comments were stripped out, but not always and not all comments. This means that there could be a small change in one of these standard contracts that could represent a vulnerability.
- The code base did not have any unit-, integration- or e2e-tests. This is worrying because we have seen vulnerabilities in many projects which do not have tests, especially in edge cases, which do not occur on the “happy” code-paths.
- There were instances where privileged roles were used to manually distribute protocol fees.

YearnRewards (1st pool) - yCurve LP tokens

The yearn pool stakes yCRV tokens from the Y pool on Curve Finance. The Y pool performs automatic yield-hunting for liquidity providers. It switches liquidity between Aave, Compound, and DyDx to provide the best yield among these platforms. Users of the yearn pool receive YFI tokens.

Source: https://etherscan.io/address/0x0001FB050Fe7312791bF6475b96569D83F695C9f

This contract is a copy of the Synthetix Unipool contract, which was reviewed by Sigma Prime in Feb 2020. Their report can be found here: https://github.com/sigp/public-audits/blob/master/synthetix/unipool/review.pdf.

The only difference is that Uniswap V1 token was changed to the yCRV token (Curve.fi yDAI/yUSDC/yUSDT/yTUSD (yDAI+yUSD...)) 0xdF5e0e81Dff6FAF3A7e52BA697820c5e32D806A8.

This contract does not have any governance logic in it.

YearnRewards (2nd pool) - Balancer BPT tokens

Pool #2 stakes pool tokens received when providing liquidity to a Balancer DAI-YFI pool. YFI is distributed to incentivize DAI-YFI liquidity. Users receive YFI tokens for providing this liquidity.

Source: https://etherscan.io/address/0x033E52f513F9B98e129381c6708F9faA2DEE5db5

This contract is a copy of the Synthetix Unipool contract, which was reviewed by Sigma Prime in Feb 2020. Their report can be found here: https://github.com/sigp/public-audits/blob/master/synthetix/unipool/review.pdf

The only difference is that Uniswap V1 token was changed to the BPT token 0x60626db611a9957C1ae4Ac5b7eDE69e24A3B76c5.

This contract does not have any governance logic in it.

YearnGovernance (3rd pool) - Balancer BPT tokens

The Governance pool stakes pool tokens received when providing liquidity to a Balancer yCRV-YFI pool. Users receive YFI, and if they stake more than 1000 pool tokens, are eligible to vote on YFI governance proposals.

Source: https://etherscan.io/address/0x3a22df48d84957f907e67f4313e3d4317904 0d6e#code

Owner and Governance are not multisig-wallet

Severity: High

- [owner] Set the reward distribution address
- [governance]
    - Transfer any token besides YFI, BPT and the reward token (`feesPaidIn`) to their own address, by calling `seize`
    - Set the breaker that will block all fee claims when withdrawing, voting and staking, by calling `setBreaker`
    - Set the reward token address
    - Set the governance address
    - Set the minimum amount needed for proposals
    - Set the number of blocks (`period`) after a proposal when one can vote
    - Set the amount of time the vote is locked

The RewardDistribution address decides what reward will be given for the next 7 days, not the governance address

Severity: Medium

The `rewardDistribution` address is not public and therefore its value cannot be read directly from the smart contract. However, looking at the transaction history of the smart contract it is visible that the `rewardDistribution` address was set in the 3rd transaction ever to be executed on this contract: https://etherscan.io/tx/0xe2b5ef06f80f3ba09992192bbcd2780d43c16da26f2e16b554d25565caa46423 and it was set to the address controlled by Andre, namely `0x2d407ddb06311396fe14d4b49da5f0471447d45c`. The `rewardDistribution` address is the only address that is allowed to call the `notifyRewardAmount` function, which can set the following state

variables:

1. `rewardRate`
2. `lastUpdateTime`
3. `periodFinish`, which is 7 days from the time when this function is called

This `notifyRewardAmount` function was called on July 19th at 03:38:14 PM +UTC in this transaction https://etherscan.io/tx/0x14b86156269 4ca6ae06cebb3416dea62c149342bf9adfe9ff28e5180082047c0 with the reward parameter set to 10,000 YFI . **This means that the `periodFinish` will be on July 26 at 03:38 PM +UTC.** At this point in time the:

- `lastTimeRewardApplicable` function will always return the `periodFinish` date, which will lead the (see next bullet)
- `rewardPerToken` function to stop increasing the reward.

Voting 'period' and 'lock' values may differ

Severity: Medium

By default the `period` and `lock` state variables in the YearnGovernance contract have the same value of 17280 blocks (approximately 3 days), which is not a problem. However, the governance address can change these values independently such that they differ. If `period` > `lock` then stakers are allowed to withdraw their stake after `lock` blocks, then they can re-stake the same amount and double their votes for the same proposal. Here are the steps to execute this attack:

1. Prerequisite: Governance address sets the `period = 18000` and leaves `lock = 17280`.
2. Alice creates a proposal (having a voting period = 18000)
3. Bob, who already has a stake of 100, places his vote. His tokens are locked for 17280 blocks.
4. After 17280 blocks, Bob's tokens are unlocked and he withdraws them.
5. Using a different address Bob places another stake using the same 100 tokens. Bob is now allowed to vote on Alice's proposal again, which increases his vote count to 200 and locks his tokens for 17280 blocks.

Recommendation: Merge the `lock` variable with the `period` variable. It's not clear why and if they will ever have different values.

A user can vote "For" and "Against" on the same proposal simultaneously

Severity: Medium

Community member Klim has brought up that a user can vote for and against simultaneously. We located the code segments in `YearnGovernance`, `voteFor()`and `voteAgainst()` functions and have confirmed the issue. While the functions check that if the user has voted for the same position before and accounts for the votes, they do not check if a user's vote has been accounted for the other stance and thus allows double voting. The issue is present in both functions.

Recommendation: We recommend adding a requirement statement and only allow the user to vote for one given stance.

YearnFeeRewards (4th pool) - stakes YFI for % of protocol fees

The Fee Rewards pool allows users who have staked more than 1000 pool tokens in Pool #3 and also voted on a proposal to stake their YFI. By staking YFI, they receive rewards in the form of yCRV tokens.

Source: https://etherscan.io/address/0xb01419E74D8a2abb1bbAD82925b19c36C191A701#code

Owner and Governance are not multisig-wallet

Severity: Medium

Description: Similarly to pool 3 above, the current `owner` and `governance` addresses are set to `0x2d407ddb06311396fe14d4b49da5f0471447d45c`. This gives the address power to transfer tokens (other than YFI and yCRV) to itself.

Users have to voteLock recently to claim rewards

Severity: Informational

The `getReward()` function requires the user to have a certain amount of yGov staked balance and have been active in governance, i.e. to have voted recently at the time they are claiming the reward. This

requirement is identical as the ones in the `stake()` function, however, it is unclear whether this is intended as it is not documented or communicated openly.

Recommendation: confirm if this is the intended behavior of the function and document the rationale to inform the users.

YFI Contract

Source: https://etherscan.io/address/0x0bc529c00c6401aef6d220be8c6ea1667f6ad93e#code

The governance address can add any address as a minter. That minter can mint tokens arbitrarily as there is no cap.

Recommendation: Cap value for minting should be decided by governance.

YearnRewards

Source: https://etherscan.io/address/0xcc9efea3ac5df6ad6a656235ef955fbfef65b862#code

YFI Tokens can be seized by the governance (yearn:Deployer) address

Severity: Medium

A total of 10.8547533840669 7 YFI tokens were seized (transferred out the contract) by the yearn:Deployer address (0x2d407ddb06311396fe14d4b49da5f0471447d45c) on the 21st of July 2020, at 12:19 UTC. This happened in 2 consecutive calls to the `seize` function shown in the 2 transactions below:

1. https://etherscan.io/tx/0x97c27ddd7b1d8c3ab006e99bcee2753df6391ff08cfbe45f81a16e93f88de9e
   From Ygov.finance: RewardsTo yearn: Deployer For 1.085475338406696829 ($2,339.54)
2. https://etherscan.io/tx/0xb2a1152ff01d7ea9da7f095c5a9b1c72e637f1569ecda528be7a01565a40c3d
   From Ygov.finance: RewardsTo yearn: Deployer For 9.769278045660271461 ($21,055.82)

The entire amount of YFI was then transferred to the DistributeYFI contract located here https://etherscan.io/address/0x812ac0eaeb422efa44eac670aa2246a25ecfa017#code

This contract has then distributed this amount non-uniformly in this transaction: https://etherscan.io/tx/0x210ad532a54eeb4fe6e007342ee72459097fb7f4e9b77ff5f090349566f4d586 to 17 addresses as indicated (hardcoded) in the `distribute()` function of the DistributeYFI contract:

1. yfi.safeTransfer(0x28b88cfD875C883cDb61938C97B8d1baabf31c88,4084730714199977);
2. yfi.safeTransfer(0x3F47A66aDA01491c3d364599e5bcBf80A1a67092,1405800000000000000);
3. yfi.safeTransfer(0x5ade40e345817B739b91c6B4615eFCE87F9D7c57,1000000000000000);
4. yfi.safeTransfer(0x6c6145d05Cd02B40403fd739f7B9B24fc8d8C410,2000000000000000);
5. yfi.safeTransfer(0xc7D2fdE79bDAe639115607A5bddd1596058E9c29,100000000000000);
6. yfi.safeTransfer(0xd6b806f51d41947B6C8465363De90941a81FD8Bd,85800000000000000);
7. yfi.safeTransfer(0x4A915B337B38f4755b39f2801D4Df2901A1432CD,75500000000000000);
8. yfi.safeTransfer(0xbA766A08f0a126Ed43B2699F722d2063e5fBb308,4282600000000000000);
9. yfi.safeTransfer(0x5398850A9399Da87624874704FEAa8A9C6C4089B,10000000000000000);
10. yfi.safeTransfer(0x93aa8C9A50EFa30Bb28cE0AD12516686c963472a,2416900000000000000);
11. yfi.safeTransfer(0xf853184415AC2312844E77Cc7BaBda372e8F56aF,257000000000000000);
12. yfi.safeTransfer(0xd53C9Fedcc95187307908D659846a443cb1e7350,122100000000000000);
13. yfi.safeTransfer(0xCC1cc238f0D0de0bB82cD1F36Ea988D8EB4489AB,243200000000000000);
14. yfi.safeTransfer(0xDda8901508211dfd3a2A912fEb0b913a6558c113,1006600000000000000);
15. yfi.safeTransfer(0x1AE3A0366C8F540C9f31cfC18A23b7E4AC9D4d8D,289700000000000000);
16. yfi.safeTransfer(0xdA581d5E0c21f59E681ADe7747F477D44e4E6FD4,6346686533527683 13);
17. yfi.safeTransfer(0xbff3BdC458D94C9Ca36230bE24838e778305A954,17700000000000000);

These addresses burned these exact amounts of YFI tokens before by calling the `claim` function of the `YearnRewards` contract, for which they received aDAI (Aave interest bearing DAI) in exchange. It is not entirely clear why they were given back the YFI tokens later. Those addresses do not seem to have returned the aDAI in this transaction history: https://etherscan.io/tokentxns?a=0xcc9efea3ac5df6ad6a656235ef955fbfef65b862
We believe this is related to the fact that some manual operations still need to be performed by the yearn: Deployer admin key.

TimelockGovernance

Severity: Informational

Source: https://etherscan.io/address/0x026d4b8d693f6c446782c2c61ee357ec561dfb61#code

The `updateTargetGovernance` and `updateThisGovernance` functions are not access controlled in any way. Anyone can call these functions. The update period is hardcoded to 3 days (17280 blocks) and cannot be updated. The `addMinter` function cannot be called by `TimelockGovernance`.

APR oracle

Source: https://etherscan.io/address/0x97FF4A1b787ADe6b94cca95b61F79417c673331D#code

APR calculation sometimes takes 365 days, other times 366

Severity: Informational

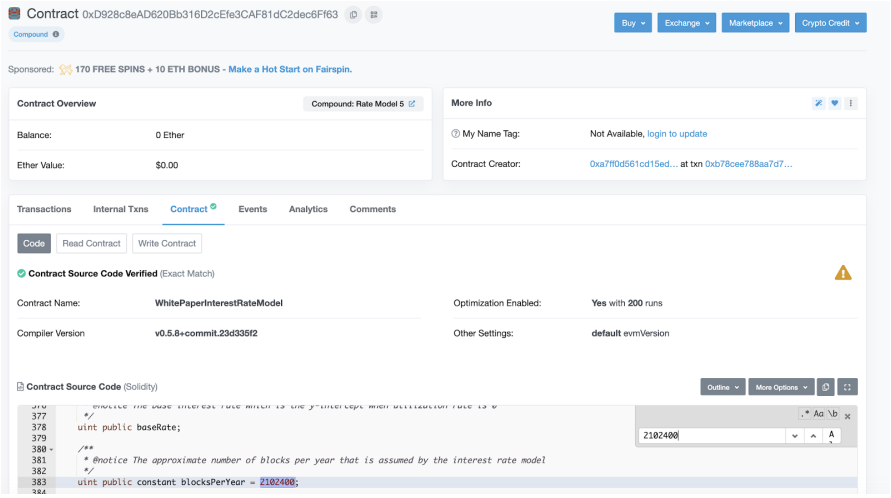
Description: For Compound, the contract calculates APR based on a 365-day calendar; for DyDx, it does so using a 366 baseline. Both are approximations, but 365 seems more accurate.

Heavy use of magic numbers

Severity: Informational

Description: The contract makes heavy use of undocumented numerical constants (magic numbers), which requires readers and auditors alike to reverse engineer the underlying context. Examples:

- In `getCompoundAPR`: 2102400. After a closer look, it stands for the blocks per year on compound:
- 1 block per ~ 15sec -> 4 blocks per minute
- 4 blocks per minute -> 240 blocks per hour
  5760 blocks per day
- 5760 blocks per day -> 2102400 blocks per year (365 days)
  The constant 2102400 is then used to calculate Compounds' APR. Note that results may differ from Compound unless the latter keeps this constant hardcoded in their contracts (currently the case). See below.



- In `getDyDxAPR`, 31622400 - stands for the number of seconds in a 366 day year.

No official documentation on DyDx APR calculation

Severity: Low

Description: DyDx does not provide any official documentation on how to calculate the supply APR for an underlying token. The developer of this APR Oracle even acknowledges that---see his Medium post "How we built on-chain APR for Ethereum DeFi". Thus, it could be the case that the current implementation does not actually match DyDx's true APR. Nonetheless, the implemented logic seems correct.

Privileged Roles - Key Person Risk

Severity: Medium/High

Description: The contract contains many restricted functions (as they should be); thus, they can only be called by the address who deployed the contract (currently, the `oracle` address, which is the same as the contract owner: 0x284b672380a4b5e4d7ccfcc0541a1d0a78c37f8c). There is a risk that if its key gets compromised, the entire Oracle becomes unusable as it could be incorrectly configured by means of its set-like functions.

Unused Functions

Severity: Undetermined

Description: The Oracle contains some functions that do not seem to be used anywhere in the given contracts we had access to. However, we cannot claim that these functions are not indeed used (e.g., it could be the case that we did not get all contracts that participate in this platform). Among those functions that are not called, we found functions that change the Oracle's price for a given token, as well as its liquidity.

No sanity check of input address parameters

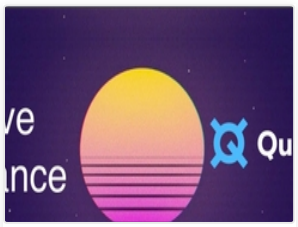
Severity: Low

Description: All set-like functions that take a contract address as a parameter do not check if the given address is different from 0x0, nor that it points to a contract. Thus, 0x0 and EOA addresses can be accidentally set.

Disclaimer

This informal security review is applicable to the current version of contracts as of July 24, 2020 10PM UTC. Quantstamp is aware of the announcement by yearn.finance on its plan to deploy v2 contracts in the next 3-4 days (https://medium.com/iearn/yearn-finance-v2-af2c6a6a3613). The v2 contracts are not yet all available and have not been included as part of this security review.

AUGUST 18, 2020



AUGUST 18, 2020

AUGUST 4, 2020

JULY 21, 2020

Quantstamp Enhancing the Securit...

Quantstamp provides security services to Layer 1...

Quantstamp Audits Curve Finance...

Quantstamp is happy to announce that we have finished our...

Quantstamp Community Update -...

Here's what happened at Quantstamp in July:

Risks on the Farm - How to Yield Far...

"Yield Farming" is on the rise. Users are making money simply by...

Leaders in Blockchain Security and Solutions

SMART CONTRACT AUDITS

- Security
- Network
- Blockchain
- Security
- Auditing
- Enterprise
- Solutions

INFORMATION

- Run a Node
- Media
- Resources
- Smart Contract
- Security Alliance

COMPANY

- Our Team
- Careers
- Contact

STAY INFORMED

Disclosure Terms and Conditions Privacy Policy
© 2017-2020 Quantstamp, Inc.