yAcademy veYFI review

Review Resources:
None provided beyond the code repository

Residents:

- NibblerExpress
- Engin33r

Table of Contents

tags: `Review` `yAcademy`

Review Summary

veYFI

The purpose of veYFI is to transition the YFI token to the vote escrow (ve) tokenomics model. The Yearn multisig will manage the contracts, which can be upgrade and migrated as needed. The GaugeFactory contract creates new Gauge and ExtraReward contracts for each Yearn Vault. The Registry contract maintains a list of active gauges and rewards pools with their corresponding vaults, which enables upgrades to the system. The Gauge contract handles deposits of YFI while the VotingEscrow.vy contract, forked and modified from Curve's ve tokenomics design, handles deposits of veYFI to enable governance rights. Some differences from Curve's ve tokenomics include the ability to withdraw locked veYFI with a penalty and the ability for locked veYFI holders to delegate votes.

The main branch of the veYFI Repo was reviewed over 20 days, 3 of which were used to create an initial overview of the contract. The code review was performed between March 28 and April 16, 2022. The code was reviewed by 2 residents for a total of 58 man hours (engn33r: 34 hours, and NibblerExpress: 24 hours). The repository was under active development during the review, but the review was limited to one specific commit.

Scope

Code Repo
Commit

The commit reviewed was 7846291efec36638eae32ac8798544050ca20877. The review covered the entire repository at this specific commit but focused on the contracts directory.

The review was a time-limited review to provide rapid feedback on potential vulnerabilities. The review was not a full audit. The review did not explore all potential attack vectors or areas of vulnerability and may not have identified all potential issues.

yAcademy and the residents make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAcademy and the residents do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. Yearn and third parties should use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	The onlyOwner modifier was applied on functions that set key protocol state variables in Registry.sol and BaseGauge.sol. Elsewhere, msg.sender is used so that the user can only control their own deposits. Access controls are applied where needed.
Mathematics	Average	Solidity 0.8.13 is used, which provides overflow and underflow protect. No unchecked code exists and no low-level bitwise operations are performed. There was no unusually complex math, except perhaps some of the vyper code functions borrowed from Curve which were not modified.
Complexity	Low	The inheritance structure, duplicate function names, lack of clear comments, and inclusion of vyper code all together can make the code hard to follow at times. There is a lack of clarity around the off-chain voting process, and no comments in the code or repository issues/PRs clarify how the off-chain voting expected to integrate with on-chain voting variables.
Libraries	Good	Only basic Open Zeppelin contracts such as SafeERC20, Ownable, and Math are imported, no other external libraries are used. Fewer and simpler external dependencies is always a plus for security.
Decentralization	Average	The onlyOwner modifier indicates there is some centralization risk, but if the owner is a Yearn Finance multisig, the risk is reduced. Contracts such as VotingEscrow.vy support migration, so some form of ownership is required.
Code stability	Average	Changes were reviewed at a specific commit and the scope was not expanded after the review was started. However, development was ongoing when the review was performed so the code was not fully frozen, which means deployed code may vary from what was reviewed.
Documentation	Low	Comments existed in many places, but were lacking in key areas. As one example, identically named _updateReward functions existed in Gauge.sol, ExtraReward.sol and VeYfiRewards.sol, but no comments existed on either function and no explanation of the differences of these identically-named function existed. It would be best if more thorough comments and documentation was added throughout the code to better explain the purpose of different functions and specific math that is performed. No developer documentation like gitbooks was observed for veYFI at the time of review.
Monitoring	Good	Events were added to all important functions that modified state variables.
Testing and verification	Average	Brownie test coverage was passing with some warnings at the commit reviewed. Test coverage was above 80% in most contract functions but not all. There was a lack of testing integration with snapshot.org voting contracts and off-chain voting strategies.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements,
- Gas savings
  - Findings that can improve the gas efficiency of the contracts
- Informational
  - Findings including recommendations and best practices

High Findings

1. High - Incompatible Vote Delegation (engn33r)

PR #99 moved veYFI to off-chain voting. Discussions with the veYFI team confirmed that the https://snapshot.org/ platform will be used for off-chain voting. The snapshot documentation delegation documentation states the `setDelegate()` function should be used when a contract is delegating votes. The VoteDelegation.sol contract implements an independent vote delegation system, and there is no call in veYFI to call the `setDelegate()` or `clearDelegate()` functions in the snapshot.org delegation contract to be compatible with snapshot.org. This means the veYFI delegation code will have no impact on snapshot.org voting.

Proof of concept

The VoteDelegation.sol contract has functions including `delegate()` and `removeDelegation()`, but the VoteDelegation.sol contract does not have any interaction with the snapshot DelegateRegistry contract. Because there are no contract calls to snapshot.org contracts, snapshot.org will not register any vote delegation.

Impact

High, vote delegation implementation is not compatible with intended offline voting approach

Recommendation

Overall the change to off-chain voting requires additional changes and integration effort. Some of the steps required to better integrate VoteDelegation.sol with snapshot.org include:

1. Creating a veYFI space in snapshot.org, which requires an ENS domain as documented in: https://docs.snapshot.org/spaces/create
2. Adding calls to snapshot.org's existing DelegateRegistry contract function calls in several places. An example implementation from Convex is provided in the documentation: https://docs.snapshot.org/guides/delegation#with-a-smart-contract. For starters, `DelegateRegistry.setDelegate()` should be called in the VoteDelegation.sol `_delegate()` function and `DelegateRegistry.clearDelegate()` should be called in the `_removeOldDelegation()` function. Additional changes may also be needed.
3. Confirm that unit tests exists that interact with the DelegateRegistry contract on the proper testnet
4. It is not immediately clear whether the erc20-balance-of snapshot.org voting strategy works with the custom vote delegation design. This should be tested before production.

I see three different approaches to handle the off-chain voting:

1. Remove vote delegation to keep everything like Curve (most simple)
2. Remove the "until" value in the vote delegation and modify the voting delegation to use the DelegateRegistry from snapshot.org
3. Implement a custom voting strategy on snapshot.org to handle the delegation and "until" values (most complex)

Medium Findings

1. Medium - Lock 1e-18 YFI to Get Rewards (NibblerExpress)

`_updateReward` computes a penalty based on the locking ratio of the account. A maximum possible lock of the smallest possible value allows a user to avoid any penalty without locking any amount of value.

Proof of concept

```
def test_gauge_cheat_yfi_lock(
    yfi, ve_yfi, whale, whale_amount, shark, shark_amount, fish, fish_amount, create_vault, create_gauge, gov
):

    yfi.approve(ve_yfi, shark_amount, {"from": shark})
    ve_yfi.create_lock(
        shark_amount, chain.time() + 4 * 3600 * 24 * 365, {"from": shark}
    )
    assert yfi.balanceOf(shark) == 0

    yfi.approve(ve_yfi, fish_amount, {"from": fish})
    ve_yfi.create_lock(
        fish_amount, chain.time() + 4 * 3600 * 24 * 365, {"from": fish}
    )
    assert yfi.balanceOf(fish) == 0

    yfi.approve(ve_yfi, 1, {"from": whale})
    ve_yfi.create_lock(
        1, chain.time() + 4 * 3600 * 24 * 365, {"from": whale}
    )
    assert yfi.balanceOf(whale) == whale_amount - 1

    lp_amount = 10**18
    vault = create_vault()
    tx = create_gauge(vault)
    gauge = Gauge.at(tx.events["GaugeCreated"]["gauge"])

    vault.mint(shark, lp_amount/100)
    vault.approve(gauge, lp_amount, {"from": shark})
    gauge.deposit({"from": shark})

    vault.mint(fish, lp_amount/100)
    vault.approve(gauge, lp_amount, {"from": fish})
    gauge.deposit({"from": fish})

    vault.mint(whale, 100*lp_amount)
    vault.approve(gauge, 100*lp_amount, {"from": whale})
    gauge.deposit({"from": whale})

    yfi_to_distribute = 10**16
    yfi.mint(gov, yfi_to_distribute)
```

```
    yfi.approve(gauge, yfi_to_distribute, {"from": gov})

    gauge.queueNewRewards(yfi_to_distribute, {"from": gov})
    assert pytest.approx(gauge.rewardRate()) == yfi_to_distribute / (7 * 24 * 3600)

    chain.sleep(3600*24*7)
    chain.mine()

    assert gauge.earned(whale) < .39 * yfi_to_distribute
    gauge.getReward({"from": whale})

    assert yfi.balanceOf(whale) < .39 * yfi_to_distribute + whale_amount

    gauge.withdraw({"from": whale})
    assert vault.balanceOf(whale) == 100*lp_amount
```

Impact

Medium - The result seems contrary from intent, but the user cannot flash loan their deposit. The user does need to keep the deposit in for seven days to reap the rewards.

Recommendation

One possible solution is to include a veYFI balance requirement when determining the penalty.

2. Medium - Flash Loan Sybil Attack to Boost Rewards (NibblerExpress)

The boosted balance for rewards is computed as `balance * 0.4 + totalSupply * veYFIBalance / veYFITotalSypply * 0.6`. (Typo in veYFITotalSupply is copied from comment in L284 of Gauge.) Although depositing in your own account calls updateReward for your account, you can use another account to flash loan a large amount of vault tokens and deposit them to boost the totalSupply. Once you have boosted totalSupply, you trigger updateReward for the target account and then withdraw the flash loan deposit.

(Note that the inverse is also true. You can either sandwich attack someone else's update by withdrawing tokens before their update to reduce total supply, or you can withdraw tokens and use getRewardFor to force an update to someone else's rewards like discussed here: https://github.com/yearn/veYFI/issues/33.)

Proof of concept

Boosted Balance equation: https://github.com/YAcademy-Residents/veYFI/blob/7846291efec36638eae32ac8798544050ca20877/contracts/Gauge.sol#L302

Deposit Update Reward call: https://github.com/YAcademy-Residents/veYFI/blob/7846291efec36638eae32ac8798544050ca20877/contracts/Gauge.sol#L351

Total Supply increase: https://github.com/YAcademy-Residents/veYFI/blob/7846291efec36638eae32ac8798544050ca20877/contracts/Gauge.sol#L362

Impact

Medium - The magnitude of the impact depends on how much you can flash loan versus what the total supply is. If you can flash loan 5x the total supply, you can get full rewards with 1/6 the intended veYFIBalance. If you can only flash loan 10% of the total supply, you only get a 9% boost to your veYFIBalance.

Recommendation

A possible solution is to compute a time weighted average of totalSupply. Alternatively, governance could set a veYFIBalance multiplier to use instead of using totalSupply/veYFITotalSupply. A third solution is to add `require(block.timestamp > lastTimeRewardApplicable());` to the `_updateReward()` function to prevent a user from performing multiple deposits/withdrawals in the same block.

3. Medium - Incorrect variables in getRewardFor call (engn33r)

The `_getReward()` function in Gauge.sol uses the wrong account address twice, which counts and distributes rewards incorrectly when account != msg.sender. After the review started, one of these variables was fixed, but not the other.

Proof of concept

The _account variable should be used in this line instead of msg.sender
https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L512

```
IExtraReward(extraRewards[i]).getRewardFor(msg.sender);
```

The _account variable should be used in this line instead of msg.sender
https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L501

```
IVotingEscrow(veToken).deposit_for(msg.sender, reward);
```

Impact

Medium, incorrect values could be calculated when account != msg.sender in some edge cases

Recommendation

Use this fix from PR #125 for both cases, which was created after the commit at which this code review started at.

4. Medium - Unclear Integration of Vote Delegation Until Value (engn33r)

This issue is similar to the high risk voting issue but relates to a difference between Curve voting (where veYFI borrows some ideas from) and the current veYFI implementation. PR #99 moved veYFI to off-chain voting. Discussions with the veYFI team confirmed that the https://snapshot.org/ platform will be used for off-chain voting. Although snapshot.org does support vote delegation, the existing snapshot.org voting delegation does not support the "until" value. This means that if the existing snapshot.org vote delegation strategy is used, adding calls to the `setDelegate()` or `clearDelegate()` functions, then delegation will remain in place until it is changed by the veYFI token owner, and can remain valid beyond the "until" timestamp.

Proof of concept

The VoteDelegation.sol contract has functions including `delegate()` and `removeDelegation()`, but the VoteDelegation.sol contract does not have any interaction with the snapshot DelegateRegistry contract. Because the actual act of voting is planned to happen off-chain with snapshot.org, snapshot.org needs to recognize what a valid vote or delegation is in the snapshot that it takes for each proposal. Currently there is no option in the snapshot.org delegation approach for an "until" value. Factoring in the "until" value could be implemented in a custom snapshot.org voting strategy, but because no such strategy is mentioned anywhere in the veYFI repository, it is expected that this has not been considered.

Impact

Medium, vote delegation implementation is not compatible with intended offline voting approach. It is not considered high risk because if the snapshot.org vote delegation is integrated (which it currently is not), the basic act of vote delegation should work and a workaround with token holders calling the right function at the right time can be used.

Recommendation

The best solution is to create a custom voting strategy, instead of using the erc20-balance-of voting strategy that Curve uses. An easier solution is to remove the "until" value in the vote delegation and use the simplistic erc20-balance-of voting strategy along with the built-in snapshot.org delegation strategy.

I see three different approaches to handle the off-chain voting:

1. Remove vote delegation to keep everything like Curve (most simple)
2. Remove the "until" value in the vote delegation and modify the voting delegation to use the DelegateRegistry from snapshot.org
3. Implement a custom voting strategy on snapshot.org to handle the delegation and "until" values (most complex)

5. Medium - `force_withdraw()` penalty may inadequately deter gamification and attacks on voting (engn33r, NibblerExpress)

The penalty ratio equation reads

```
penalty_ratio: uint256 = min(MULTIPLIER * 3 / 4, MULTIPLIER * time_left / MAXTIME)
```

Short lock periods receive low withdrawal penalties, which may not be intentional. Because the MAXTIME value is 4 years, when the time_left value is less than 3 years, the penalty ratio for withdrawal will be less than 75%. This means for any lock period under 3 years, the worst case withdrawal penalty, if the user withdraw immediately after locking, is less than 75%. This could lead to gamification where users planning to withdraw will lock the veYFI for shorter durations instead of 4 years.

The inverse scenario could also cause a problem. Users receive a 100% credit for deposits with a four year lock but only a 75% burn. If there is a scenario where the benefit of a temporary deposit is > 75% of the deposit, a user should do a max lock and then take the burn. For example, if a proposal put forth for a vote could allow a user to drain a contract, it could be profitable for a user to deposit sufficient funds to have > 50% of the voting power and to retrieve 25% of their deposit plus what can be drained from the contract.

This attack vector would most likely happen when a user has advanced knowledge that a vote is upcoming (YIP proposals and related discussion are often public). The user could stake YFI for a specific vote (either a governance or gauge vote) where they may disproportionately benefit. This vector may be unique to veYFI compared to other ve protocols because veYFI does not apply protective measures that other protocols use. Curve does not all for early withdrawal of locked veCRV, which protects against this scenario. Convex allows for vote delegation, similar to veYFI, but adds a 16 week cooldown period before the locked tokens can be used for voting, making this attack vector less likely due to the advanced planning necessary. https://docs.convexfinance.com/convexfinance/general-information/voting-and-gauge-weights/vote-locking

The risk for this vector is increased when combined with finding Low #3, because Low #3 outlines an approach that could enable the user paying the penalty to receive some of their penalty back in rewards.

Proof of concept

The penalty_ratio equation
https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VotingEscrow.vy#L570

Impact

Medium, the penalty calculation could be gameable because incentives are not properly aligned towards long-term locking of veYFI with the existing penalty calculations.

Recommendation

Consider modifying the function so that the penalty ratio is a function of the time_left to locked duration, not a function of the ratio of the time_left to 4 years. This could reduce gamification by increasing the penalty cost in this type of attack. The modified code below is a draft and should be thoroughly tested, but it is intended to provide results of:

- If a user hypothetically locks for 4 years and withdraws in their first year, 75% penalty (same as before)
- If a user hypothetically locks for 1 year and withdraws immediately, 75% penalty. With the old equation, the penalty was closer to 25%

```
penalty_ratio: uint256 = min(MULTIPLIER * 3 / 4, MULTIPLIER * time_left / (block.timestamp -
user_point_history__ts(msg.sender, self.user_point_epoch[msg.sender])))
```

Require tokens to be locked for a certain time before (and possibly after) allowing voting, similar to Convex. Increasing the ease of withdrawal for locked tokens, even with a penalty, can add unexpected incentives.

Low Findings

1. Low - Voting delegation is missing edge case checks (engn33r)

The VoteDelegation.sol contract permits vote delegation. The delegation has an "until" value, which can be set to zero. The until value can be changed so that the new until value is greater than the old one, but the check should also confirm that the new until value is greater than block.timestamp. Otherwise the until value can be modified for a time in the past. This may cause issues in the voting logic depending how the until value is stored and used in the snapshot logic.

Proof of concept

The VoteDelegation.sol contract permits an "until" variable value of zero if the tokens are not locked. Any "until" variable value that is less than block.timestamp should have no impact on voting or delegation. But if the "until" value starts at zero, it is possible for the value to be increased to a value less than block.timestamp to reference a time in the recent past, but still have no impact. It is possible that the voting logic, which is not within the scope of these solidity contracts, may not properly consider this edge case. This edge case can be solved by adding a check in all functions that modify the "until" state variable, including `_delegate()` and `increaseDelegationDuration()`, to validate `require(until > block.timestamp)`. An exception could be made for an until value of zero, resulting in a require statement that looks like `require(_until > block.timestamp || _until == 0)`

Another edge case is a situation where the until value is set to a time beyond the lock expiration of the veYFI tokens. There is no check for to confirm `until < locked.end`. This edge case could also happen if a user delegated votes, then withdrew their tokens to a new account where the tokens were locked and votes delegated again. It is unclear whether the off-chain voting mechanism will handle this edge case, which could allow multiple votes from a single token. Otherwise, consider adding the check `require(_until < locked.end)`

Impact

Low, risk may be greater depending on the off-chain voting logic

Recommendation

Add logic checks into the VoteDelegation.sol contract to prevent mistakes with the off-chain voting logic that results in unexpected edge cases

2. Low - Sawtooth Wave Effect from Computing veYFI Balance and Supply Using Different Time Scales (NibblerExpress)

The boosted balance for rewards is computed as `balance * 0.4 + totalSupply * veYFIBalance / veYFITotalSypply * 0.6`. (Typo in veYFITotalSupply is copied from comment in L284 of Gauge.) Most voting escrow functions truncate time stamps down to the most recent week. This includes the function used to compute total supply. The veYFIBalance is not truncated to the nearest week. A user with a max lock will see their balance slowly decline over the course of a week until they can increase the lock again when the next week arrives. Because anyone can update the rewards for a user (e.g., using depositFor or getRewardsFor), an attacker can wait until the very end of a week when veYFIBalance is lowest to update the rewards for other users.

Proof of concept

Equation to compute Total Supply: https://github.com/YAcademy-Residents/veYFI/blob/7846291efec36638eae32ac8798544050ca20877/contracts/VotingEscrow.vy#L756
(`t_i` is truncated to a value in weeks at L748 and L750.)

Equation to compute Balance: https://github.com/YAcademy-Residents/veYFI/blob/7846291efec36638eae32ac8798544050ca20877/contracts/VotingEscrow.vy#L677

`_t` is set to block.timestamp for balance computation: https://github.com/YAcademy-Residents/veYFI/blob/7846291efec36638eae32ac8798544050ca20877/contracts/VotingEscrow.vy#L647

Impact

Low - The decline over the course of a week will be < 0.5%, so users can boost their veYFI balance by 0.5% to avoid any attack. Also, the attack likely won't be worth the cost of gas. This may be a small annoyance for users depositing amounts close to the cutoff for full rewards.

Recommendation

The balance calculation could be truncated to the nearest week like the calculations in Total Supply and other functions.

3. Low - Equal penalty reward distribution not incentive aligned (engn33r, NibblerExpress)

When a user withdraws the locked veYFI from VotingEscrow.vy early, the penalty fee is redistributed in the rewards pool for that week. The rewards are distributed evenly to all reward recipients, and are not distributed by the veYFI weights or locking period of each veYFI holder. There is an incentive to actively deposit and withdraw after a penalty fee is received by the veYFI protocol while the rewardPerToken is increased. This can lead to backrunning opportunities when a penalty fee is paid.

There is another penalty fee applied in Gauge.sol when `_updateReward()` is called using the updateReward modifier. This penalty will be non-zero any time that the tokens are not locked for 4 years. If a user was going to pay the Gauge.sol penalty and was looking to extract some of their penalty payment, they can plan ahead to stay under the 120% limit imposed by `queueNewRewards()` in BaseGauge.sol.

This is slightly similar to this Solidly issue, but this veYFI issue only impacts penalty fee payouts, not gauge reward payouts: https://github.com/belbix/solidly/issues/1

Proof of concept

A discussion for this issue was started by someone from Ribbon Finance (who forked the veYFI code) before the review ended in veYFI Issue 135 and a fix is being developed in PR #136. The issue is centered around how penalty fees paid in VotingEscrow.vy and Gauge.sol are redistributed to users.

Impact

Low, the rewards payout from a penalty is not well distributed, but users are incentivized to lock for MAX_TIME and not to withdraw early to avoid penalties.

Recommendation

Improve the reward distribution of penalty fees, which is a difficult problem to resolve. Paying out penalty fees slowly over a longer time period instead of adding the funds to the current epoch reward pool would reduce the chances that a large amount of penalty fees could be extracted quickly.

4. Low - No penalty during migration (engn33r)

The `_lockingRatio()` function of Gauge.sol returns a ratio based on the amount of time the user has locked their ve tokens. The maximum value for this ratio is PRECISON_FACTOR, which is when a user locks their ve tokens for MAX_TIME. A user can also have a PRECISON_FACTOR locking ratio when the ve token contract is undergoing migration and the contract owner has not call `setVe()` yet to point to the new ve token contract. A user can take advantage of this edge case. For example, any function with the updateReward modifier will calculate a penalty of zero when a ve token migration is ongoing, regardless of how long a user has locked their tokens.

Proof of concept

The `_lockingRatio()` function returns PRECISON_FACTOR, equivalent to locking for the MAX_TIME value, when the ve token is undergoing migration:
https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L221

Instead, it would be better the revert in this case so that the ve token can be finished without any users taking advantage of this edge case.

Impact

Low, this is a rare edge case but should be handled properly

Recommendation

Change the behavior of the `_lockingRatio()` function to revert during migration to allow the owner to call `setVe()` with the new ve token address.

Gas Savings Findings

1. Gas - Using "unchecked" (engn33r)

In `setDuration()` of BaseGauge.sol, we know `periodFinish > block.timestamp` so we can use the "unchecked" clause for gas savings.

The same improvement can be used in Gauge.sol

Proof of concept

The relevant code block

```
if (block.timestamp < periodFinish) {
    uint256 remaining = periodFinish - block.timestamp;
```

Applying unchecked for gas savings, this code would look like

```
if (block.timestamp < periodFinish) {
    unchecked { uint256 remaining = periodFinish - block.timestamp; }
```

BaseGauge.sol line that can be unchecked
https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/BaseGauge.sol#L79

Gauge.sol line that can be unchecked
https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L229

Impact

Gas savings

Recommendation

Use unchecked where there is no overflow or underflow risk

2. Gas - Using simple comparison (engn33r)

Using a compound comparison such as ≥ or ≤ uses more gas than a simple comparison check like >, <, or ==. Compound comparison operators can be replaced with simple ones for gas savings.

Proof of concept

The `_notifyRewardAmount()` function in BaseGauge.sol contains
https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/BaseGauge.sol#L170-L177

```
if (block.timestamp >= periodFinish) {
    rewardRate = reward / duration;
} else {
    uint256 remaining = periodFinish - block.timestamp;
    uint256 leftover = remaining * rewardRate;
    reward = reward + leftover;
    rewardRate = reward / duration;
}
```

By switching around the if/else clauses, we can replace the compound operator with a simple one

```
if (block.timestamp < periodFinish) {
    uint256 remaining = periodFinish - block.timestamp;
    uint256 leftover = remaining * rewardRate;
    reward = reward + leftover;
    rewardRate = reward / duration;
} else {
    rewardRate = reward / duration;
}
```

Impact

Gas savings

Recommendation

Replace compound comparison operators with simple ones for gas savings

3. Gas - Use prefix in loops (engn33r)

Using a prefix increment (++i) instead of a postfix increment (i++) saves gas for each loop cycle and so can have a big gas impact when the loop executes on a large number of elements.

The gas savings comes from the removal of a temporary variable. j++ is equal to 1 but j equals 2, while in ++j both j and ++j equal 2.

Proof of concept

There are 4 instances of this in Gauge.sol and 1 instance in VoteDelegation.sol

- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L161
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L357
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L386
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L511
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VoteDelegation.sol#L121

Impact

Gas savings

Recommendation

Increment with prefix addition and not postfix in for loops

4. Gas - Payable functions can save gas (engn33r)

If there is no risk of a function accidentally receiving ether, such as a function with the onlyOwner modifier, this function can use the payable modifier to save gas.

Proof of concept

The following functions have the onlyOwner modifier and can be marked as payable

- `setVe()` https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Registry.sol#L49
- `addVaultToRewards()` https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Registry.sol#L67
- `removeVaultFromRewards()` https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Registry.sol#L94
- `setVe()` https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VeYfiRewards.sol#L30
- `setVe()` https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L110
- `setDuration()` https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/BaseGauge.sol#L73
- `sweep()` https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/BaseGauge.sol#L111

Impact

Gas savings

Recommendation

Mark functions that have onlyOwner as payable for gas savings. This might not be aesthetically pleasing, but it works

5. Gas - Use != 0 for gas savings (engn33r)

Using `> 0` is more gas efficient than using `!= 0` when comparing a uint to zero. This improvement does not apply to int values, which can store values below zero.

Proof of Concept

Four instances of this were found

- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L353
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L382
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L497
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/ExtraReward.sol#L123

Impact

Gas savings

Recommendation

Replace `> 0` with `!= 0` to save gas

6. Gas - Use Solidity errors in 0.8.4+ (engn33r)

Using solidity errors is a new and more gas efficient way to revert on failure states

- https://blog.soliditylang.org/2021/04/21/custom-errors/
- https://twitter.com/PatrickAlphaC/status/1505197417884528640

Proof of Concept

Require statements are used throughout the contracts and error messages are not used anywhere. Using this new solidity feature can provide gas savings on revert conditions.

Impact

Gas savings

Recommendation

Add errors to replace each `require()` with `revert errorName()` for greater gas efficiency

7. Gas - Declare functions external for gas savings (engn33r)

The boostedBalanceOf function and deposit function of Gauge.sol should be declared external, not public, for gas savings

Proof of concept

There are two public functions that can be external

- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L287
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L317

Impact

Gas savings

Recommendation

Declare functions as an external functions for gas savings

8. Gas - Remove Aragon calls for gas savings (engn33r)

The VotingEscrow.vy contract contains some functions that primarily serve as compatibility interfaces when using Aragon. If Aragon compatibility is not needed, these functions can be removed.

Proof of concept

The following functions include comments indicating they are fulling or partially used for Aragon compatibility, and may not otherwise be necessary:

1. `version` state variable and `_version` init parameter https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VotingEscrow.vy#L155
2. `controller()` https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VotingEscrow.vy#L129-L131
3. `transfersEnabled()` https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VotingEscrow.vy#L129-L131
4. `balanceOf()` https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VotingEscrow.vy#L650
5. `totalSupply()` https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VotingEscrow.vy#L772
6. `changeController()` https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VotingEscrow.vy#L831-L837

Impact

Gas savings

Recommendation

Remove unnecessary functionality copied from the original Curve contract for gas savings. This was mentioned to veYFI devs while the review was ongoing and a fix was carried out before the review concluded in PR 131, although the `balanceOf()` and `totalSupply()` functions were left unchanged.

9. Gas - Inconsistent zero case in VotingEscrow.vy (engn33r)

The VotingEscrow.vy contract has two similar functions, `balanceOfAt()` and `balanceOf()`, which have been modified from the original Curve contract. The upoint.bias zero case is handled differently in the two functions. Using the lower gas solution in both would be better.

Proof of concept

The `balanceOf()` function uses this code:

```
if upoint.bias < 0:
    upoint.bias = 0
    return   convert(upoint.bias, uint256)
```

while the `balanceOfAt()` function uses this code:

```
if upoint.bias >= 0:
    return convert(upoint.bias, uint256)
else:
    return 0
```

Impact

Gas savings

Recommendation

Use the `balanceOfAt()` function's if/else code in `balanceOf()` to avoid calling `convert()` for the zero case.

10. Gas - Remove duplicated code (engn33r)

The BaseGauge.sol `queueNewRewards()` function contains duplicated code. This can be simplified to save as on deployment and reduce code complexity. There might be slightly more gas spent on function calls where `block.timestamp >= periodFinish` because unnecessary math for elapsedSinceBeginingOfPeriod and distributedSoFar will be performed in this case, but deployment gas may be important because the BaseGauge.sol contract is inherited in many places.

Proof of concept

The logic for the case `block.timestamp >= periodFinish` and `(distributedSoFar * 12) / 10 < _amount` is the same, so they can be placed into the same if statement.

- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/BaseGauge.sol#L146-L150
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/BaseGauge.sol#L156-L162

The revised `queueNewRewards()` function can look like this

```
function queueNewRewards(uint256 _amount) external override returns (bool) {
    require(_amount != 0, "==0");
    SafeERC20.safeTransferFrom(
        IERC20(rewardToken),
        msg.sender,
        address(this),
        _amount
    );
    emit RewardsQueued(msg.sender, _amount);
    _amount = _amount + queuedRewards;

    uint256 elapsedSinceBeginingOfPeriod = block.timestamp -
        (periodFinish - duration);
    uint256 distributedSoFar = elapsedSinceBeginingOfPeriod * rewardRate;
    // we only restart a new week if _amount is 120% of distributedSoFar.

    if (block.timestamp >= periodFinish || (distributedSoFar * 12) / 10 < _amount) {
        _notifyRewardAmount(_amount);
        queuedRewards = 0;
    } else {
        queuedRewards = _amount;
    }
    return true;
}
```

Impact

Gas savings

Recommendation

Use simplified code from above.

11. Gas - SLOAD gas savings (engn33r)

The BaseGauge.sol `setDuration()` function can use a minor gas savings.

Proof of concept

Existing code
https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/BaseGauge.sol#L80-L84

Modified code to use local variable instead of state variable in emit

```
if (block.timestamp < periodFinish) {
    uint256 remaining = periodFinish - block.timestamp;
    uint256 newRate = remaining * rewardRate / newDuration;
    rewardRate = leftover;
}
duration = newDuration;
emit DurationUpdated(newDuration, newRate);
```

A similar type of improvement can be made in Registry.sol
https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Registry.sol#L95-L96
The code can be modified to

```
address gauge = gauges[_vault];
require(gauge != address(0x0), "!exist");
```

Impact

Gas savings

Recommendation

Use modified code snippets above.

Informational Findings

1. Informational - SafeERC20 functions not used (engn33r)

The ERC20 `approve()` function is used several times, but there can be security benefits to using the `safeIncreaseAllowance()` and `safeDecreaseAllowance()` functions instead to prevent double spend attacks. This improvement was noted in issue 69 but was not properly applied.

Proof of concept

Gauge.sol line 500 => rewardToken.approve(address(veToken), reward);
Gauge.sol line 527 => IERC20(rewardToken).approve(veYfiRewardPool, toTransfer);
VeYfiRewards.sol line 140 => rewardToken.approve(address(veToken), reward);

Impact

Informational, the return value is not checked so it is possible that an error occurs and the code doesn't revert when it should. The risk may be elevated if the rewardToken is an arbitrary ERC20 token.

Recommendation

Use the `safeIncreaseAllowance()` and `safeDecreaseAllowance()` functions if there is a possibility that the ERC20 tokens getting approved may not revert when an approve call fails.

2. Informational - Unspecified Voting Requirements (engn33r)

Voting on snapshot.org requires a voting strategy to be selected for each proposal, and no voting strategy is chosen for veYFI yet. veYFI is borrowing significantly from Curve and may use the same erc20-balance-of voting strategy, but there are some differences between veYFI and Curve (veYFI supports vote delegation, Curve does not) and some of the veYFI contract code makes assumptions about how the off-chain voting strategy will work. If these contract assumptions do not match how the voting strategy actually works, the off-chain voting process may not align with how the on-chain code intends.

Proof of concept

There are built-in voting assumptions in the contract code, mostly borrowed from Convex:

1. When a proposal is created, the `checkpoint()` function in VotingEscrow.vy should be called to properly set the epoch
2. Voting power should be calculated using the on-chain functions such as `get_last_user_slope()`, `balanceOf()`, `balanceOfAt()`, `supplyAt()`, `totalSupply()`, and `totalSupplyAt()` in VotingEscrow.vy. Note that some of the math in these calculations could be moved to an off-chain voting strategy for gas savings
3. Voting for a proposal with veYFI is allowed as soon as the veYFI is locked. Because there is potential for gamification in increasing veYFI stake shortly before key votes,
   protocols like Convex have implemented a minimum locking duration of several weeks before a token holder can vote, which is implemented in their VotingEligibility.sol contract.

Some open and unanswered questions include:

1. Who is allowed to create proposals on snapshot.org
2. How are the results of each snapshot vote converted to on-chain rewards? Is this automated?
3. Will all vaults receive a gauge that can be voted for on snapshot.org, even those holding very little value? If so, this could lead to gamification of deprecated or old vaults.
4. Is there a maximum allocation a gauge can receive? Is there minimum quorum for governance votes? Convex specifies these values: http

s://docs.convexfinance.com/convexfinance/general-information/voting-and-gauge-weights

Impact

Informational, because the off-chain voting is heavily interconnected with the on-chain contracts and unresolved questions leave risk around implementation errors in the off-chain voting proposal or strategy process

Recommendation

Several items should be considered:

1. Choose an off-chain voting strategy from snapshot.org and outline all the assumptions for the voting process. veYFI is borrowing heavily from Convex, but using the same erc20-balance-of snapshot.org strategy should be thoroughly analyzed given the protocol and liquidity differences between Yearn Finance and Curve.
2. Determine whether the gamification risks that Convex attempts to prevent with their voting caps and voting eligibility requirements are problematic enough to add similar modifictions to veYFI
3. Consider automatic on-chain execution strategies to reduce centralization risk as outlined in: https://blog.aragon.org/snapshot/

3. Informational - Variable naming inconsistency (engn33r)

The first input to the `createGauge()` is named vault, and is passed to Gauge.sol `initialize()`. In `initialize()`, this value is named `_stakingToken`. If the goal is to allow gauges to correspond to liquidity pools that are NOT yearn vaults, the naming should be consistent through veYFI, otherwise future edits may be made assuming that only yearn vaults are involved. This was mentioned in a comment in issue 46.

Proof of concept

The first input to `initialize()` has the following comment
https://github.com/yearn/veYFI/blob/d53465c5f615a04204faed55728840a1e79377fc/contracts/Gauge.sol#L67
`_stakingToken The vault token to stake`

Elsewhere in veYFI, the naming convention assumes that each gauge corresponds to one Yearn vault. But the name "_stakingToken" and the comment in issue 46 indicates this might not be correct. This should be clarified in the comments and the variable names.

Impact

Informational

Recommendation

If the intention is to limit the value of this input to only permit Yearn vaults, consider verifying whether the vault is in the list of addresses from the `assetsAddresses()` function of the AddressesGeneratorV2Vaults contract
https://github.com/yearn/yearn-security/blob/master/SECURITY.md#production-contracts

Otherwise, if this value will not be limited to Yearn vaults, consider renaming the variables that use the word "vault" or add comments in multiple places to clarify this address may not be a yearn vault.

4. Informational - Missing 0 check in `setDuration()` (engn33r)

There is no zero check in `setDuration()` in BaseGauge.sol. If a duration of zero is chosen, functions like `_notifyRewardAmount()` that divide by the duration will revert with a divide by zero error. Zero checks exist on all similar setter functions.

Proof of concept

If `block.timestamp < periodFinish`, then the code will check if newDuration is zero, but this check will not happen if the if statement is skipped when `block.timestamp > periodFinish`
https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/BaseGauge.sol#L73

This can cause problems in other functions that divide by the duration
https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/BaseGauge.sol#L170-L177

Impact

Informational, if the duration is set to zero the owner can change the value to a new non-zero value later

Recommendation

Add a zero check in the `setDuration()` function with `require()`

5. Informational - Typos (engn33r)

There are some typos that have no impact on code functionality, but fixes could be considered improvements

Proof of concept

1. The MAX_DELAGATED variable should be spelled MAX_DELEGATED
   https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VoteDelegation.sol#L16
2. betwwen should be between
   https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L324
3. The last sentence in the ExtraReward.sol contract needs to be fixed, most likely by removing the words "Gauge will". It reads "Gauge will this contract is used behind multiple delegate proxies."
   https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/ExtraReward.sol#L13-L14
4. "claimm veYFI and aditional reward" should be "claim veYFI and additional reward" (two typos)
   https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L373
5. Triger should be Trigger
   https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/BaseGauge.sol#L131
6. veYFITotalSypply should be veYFITotalSupply
   https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L284
7. "rewards are distributed during 7 days" should says "rewards are distributed during reward duration" because duration is a variable and could change
   https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/BaseGauge.sol#L12
8. "over a week" should say "over the reward duration" because duration is a variable and could change
   https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/BaseGauge.sol#L130
9. "restart a new week" should say "restart a new reward duration" because duration is a variable and could change
   https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/BaseGauge.sol#L154
10. A reference to CRV exists in the vyper code. Change ERC20CRV to ERC20YFI
    https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VotingEscrow.vy#L152
11. "earning for an account" should be "earnings for an account"
    https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L248-L249
12. "address acccount" should be "address account"
    https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L219

Impact

Informational

Recommendation

Fix typos

6. Informational - Replace magic numbers with constants (engn33r)

Constant variables should be used in place of magic numbers to prevent typos. The magic number 1e18 is found in multiple places in YieldStreamer.sol and should be replaced with a constant. Using a constant also adds a description using the variable name to explain what this value is for.

Proof of concept

The value 1e18 appears throughout the contracts

- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VeYfiRewards.sol#L62
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VeYfiRewards.sol#L69
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L245
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L274
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L280
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/ExtraReward.sol#L75
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/ExtraReward.sol#L86
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/ExtraReward.sol#L92

Impact

Informational

Recommendation

Use constant variables instead of magic numbers

7. Informational - Code inconsistency (engn33r)

There are three `_updateReward()` functions in veYFI. The Gauge.sol `_updateReward()` function includes a check if `(_balances[account] != 0)` while the other two functions do not. It is unclear if this check should be added to the other functions or can be removed from Gauge.sol, but the developers should compare these three implementations for differences.

Proof of concept

The three different `_updateReward()` functions

1. https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/ExtraReward.sol#L44
2. https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VeYfiRewards.sol#L36
3. https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L183

Impact

Informational

Recommendation

Use consistent logic in similar functions.

8. Informational - Curve safety check removed (engn33r)

There is a check in Curve's VotingEscrow.vy to prevent contracts from calling certain functions related to increase the amount of locked ve tokens. This is most likely to prevent a rare scenario where a perfect storm is created when:

1. A user has granted the ve token contract infinite allowance
2. The user has a large amount of tokens that they purposefully have not locked
3. The user interact with a malicious contract that acts as a proxy to lock the user's tokens in the ve token contract

If this sequence of events occurs, the user will not be able to withdraw their locked tokens without a loss of value. However, this is a minor problem compared to most interactions with malicious contracts, so it may be reasonable to remove this check and allow contract interaction with these functions.

Proof of concept

This is the check in Curve:

- https://github.com/curvefi/curve-dao-contracts/blob/master/contracts/VotingEscrow.vy#L418
- https://github.com/curvefi/curve-dao-contracts/blob/master/contracts/VotingEscrow.vy#L438
- https://github.com/curvefi/curve-dao-contracts/blob/master/contracts/VotingEscrow.vy#L455

The check does not exist in the same locations in the veYFI VotingEscrow contract:

- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VotingEscrow.vy#L458
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VotingEscrow.vy#L496
- https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VotingEscrow.vy#L512

If this check is added to veYFI, it should also be added to createLockFor, a function that Curve does not have: https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VotingEscrow.vy#L477

Impact

Informational, this check only adds value if a user interact with a malicious contract and has non-zero allowance to transfer to the veYFI contract which is an extreme edge case that doesn't necessarily need special handling

Recommendation

Consider reintroducing the `self.assert_not_contract(msg.sender)` check found in the original Curve code in the locations mentioned above.

9. Informational - No migrateLock sample implementation (engn33r)

The veYFI contract can be migrated. The migrator contract will need to implement the `migrateLock()` function, which is called by the existing implementation of veYFI. No sample implementation of this function is in veYFI, meaning that a core piece of the migration functionality was not examined in this review.

Proof of concept

The `migrateLock()` call in the existing veYFI code:
https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/VotingEscrow.vy#L606

Impact

Informational

Recommendation

Consider creating a reference implementation for the `migrateLock()` function before deploying veYFI so any last minute changes in veYFI that can make the lock migration easier can be handled in advance. This is especially important if the migrate function might be used in an emergency situation.

10. Informational - Use -= to keep code concise (engn33r)

The `withdraw()` function of Gauge.sol can use -= to reduce code complexity.

Proof of concept

The += operator is used often in veYFI, but the -= operator is not used in this line
https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L391

```
_balances[msg.sender] = _balances[msg.sender] - _amount;
```

Consider changing this line to

```
_balances[msg.sender] -= _amount;
```

Impact

Informational

Recommendation

Use -= when available for concise code.

11. Informational - Update rewards for other users (engn33r)

The `depositFor()` function in Gauge.sol allows the `_updateReward()` function to be called for other depositors. This may lead to an attack where a depositor in a gauge finds a way to reduce the rewards for other depositors of the same gauge, possibly similar to the flash loan method mentioned in Medium #3, and calling `depositFor()` with an extremely small value to trigger `_updateReward()` to update the checkpoint for the other depositors. The attacker may be able to take more rewards from the gauge than they could otherwise. This was noticed at the very end of the review so examination of an attack vector was not performed.

Proof of concept

The `depositFor()` function:
https://github.com/YAcademy-Residents/veYFI/blob/master/contracts/Gauge.sol#L344

Impact

Informational

Recommendation

If there is no strong use case for `depositFor()`, removing this function would be the safest approach.

Final remarks

engn33r

I focused on the voting mechanics because there have been bugs here on similar projects. Once the voting implementation is fully completed and reviewed, I think veYFI will look more solid. There are some minor edge cases in the rewards system that can be ironed out, but the most critical deposit, withdraw, lock, and rewards functions look like they have been thoughtfully pieced together into a cohesive solution.
Improvements to the overall documentation of the veYFI contracts and inclusion of similar protective measures to Convex are nice-to-have improvements.

NibblerExpress

I looked for ways to attack the deposit, withdraw, and reward functions to receive excess rewards or to block others from receiving rewards. I was unable to find attacks that circumvented the reward checkpointing or that manipulated the voting escrow contract. The problems seemed to be in how rewards were actually calculated. The reward formula should be reviewed to ensure that the contract is more precisely incentivizing the desired behaviors. I did not take a close look at how voting would occur outside the contract and what mischief could be caused by malicious proposals.

About yAcademy

yAcademy is an ecosystem initiative started by Yearn Finance and its ecosystem partners to bootstrap sustainable and collaborative blockchain security reviews and to nurture aspiring security talent. yAcademy includes a fellowship program and a residents program. In the fellowship program, fellows perform a series of periodic security reviews and presentations during the program. Residents are past fellows who continue to gain experience by performing security reviews of contracts submitted to yAcademy for review (such as this contract).

Appendix and FAQ

tags: `Review` `yAcademy`