yAudit Yearn 1UP Review

Review Resources:

- 1UP codebase

Auditors:

- panda
- ljmanini

Table of Contents

Review Summary

Yearn 1UP

1UP is a boost aggregator for Yearn's veYFI operating as a neutral public good. Written from the ground up, 1UP's immutable codebase allows YFI holders, Yearn contributors, and teams, to lock YFI into veYFI while at the same time aggregating and sharing the boost amongst one another. This is achieved without extracting value from the system, 100% of fees levied are directed back into the protocol for the sole purpose of perpetually growing its veYFI position, deepening liquidity, and growing the user base.

The contracts of the Yearn 1UP 1upyfi were reviewed over 11 days. The code review was performed by 2 auditors between March 11 and March 25, 2024. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit fdbe1262991eb2964cd9ae8527184b6f0cc3548e for the Yearn 1UP repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
contracts
├── BasicRedeemer.vy
├── Factory.vy
├── GaugeRewards.vy
├── Gauge.vy
├── LiquidLocker.vy
├── Proxy.vy
├── Registry.vy
├── StakingRewards.vy
└── Staking.vy
```

After the findings were presented to the Yearn 1UP team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Yearn 1UP and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	The reviewed contracts effectively manage access control, employing checks to ensure that only authorized addresses can execute sensitive functions. The implementation of roles and ownership is clear and adheres to best practices, minimizing the risk of unauthorized access.
Mathematics	Good	The contract logic incorporates robust mathematical operations ensuring accurate calculations.
Complexity	Good	The contracts maintain a manageable level of complexity, with a clear separation of concerns and functionality. Despite the sophisticated interactions between contracts, the codebase is structured in a way that facilitates understanding and limits the potential for bugs related to complex interactions.
Libraries	Good	No usage of libraries.
Decentralization	Low	The system shows a commitment to decentralization with its design and functionality. However, there are instances where centralized control points exist, such as the management role's ability to change critical contract parameters and do arbitrary calls using the proxy. This could be improved by further distributing governance responsibilities to the community via voting contracts or implementing additional checks.
Code stability	Good	The codebase demonstrates a high degree of stability. It adheres to Vyper best practices, with contracts written to handle various edge cases effectively.
Documentation	Good	The documentation is comprehensive, providing clear explanations of the contracts' functionalities, their interactions, and the rationale behind key design decisions.

Category	Mark	Description
Monitoring	Good	The system includes mechanisms for monitoring contract activity and performance, such as event logs for significant actions. While the audit identifies areas where additional events could improve transparency, the current setup provides a solid foundation for tracking and analyzing contract behavior.
Testing and verification	Average	The inability to accurately assess test coverage due to limitations within the testing framework itself presents a notable concern.

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.

Medium Findings

1. Medium - StakingRewards `harvest` calls can be sandwiched to extract profit.

The `Staking.vy` smart contract integrates a delayed withdrawal mechanism specifically designed for `LiquidLocker.vy` tokens. There's a potential opportunity to establish an AMM (Automated Market Maker) market facilitating trade between `Staking.vy` and `LiquidLocker.vy` tokens. This market could provide an alternative for users desiring immediate access to their assets, circumventing the default withdrawal waiting period. Within this ecosystem, a unique arbitrage opportunity arises, allowing arbitrageurs to divert a portion of the rewards from the `harvest()` function of the original funds' owner. This strategy involves borrowing `Staking.vy` tokens just before executing a `harvest()` operation. After completing the `harvest()`, the acquired rewards are swiftly returned.

Furthermore, the creation of a borrowing market like Ajna market would permit the borrowing of unstaked 1upYFI tokens, which could be deposited before a harvest reducing reward yields for every staked 1upYFI. The tokens are locked for a week, but this method bears no risk of adverse price movements for the attacker since the 1upYFI token is borrowed and can be repaid as soon as sufficient streamed assets become available.

Technical Details

The contract architecture allows the `report()` function to operate independently of a prior `harvest()` operation for token transfers or deposits. This design enables immediate reward claims following a `harvest()` operation, facilitating reward extraction in a single transaction.

References:

- StakingRewards.vy#L244
- StakingRewards.vy#L200

Impact

Medium.

1. With the introduction of an AMM, the original funds will not receive rewards.
2. The addition of more staked tokens reduces the reward per token stake.

Recommendation

Implementing a time-based reward distribution mechanism could mitigate this issue.

Developer Response

Regarding the impact:

1. By LPing into a upYFI/supYFI AMM pool you forfeit the staking rewards, as they would accrue to the AMM instead. Even if the AMM redistributed those rewards to its LPs, it is considered acceptable to us that those rewards get arbed away, as we do not want to encourage supYFI liquidity on AMMs.

2. While it is true that in a lending market a borrower could borrow upYFI, frontrun a harvest with a stake and claim rewards, this is acceptable and in practice should not negatively impact other staker rewards. The reason for that is because a lender that chooses to lend out upYFI could have used those tokens to stake instead. The lender will want to make sure that the interest they receive from lending out their tokens at least makes up for their lost potential staking rewards. Therefore, in an efficient market it should not be hugely profitable for an arber to pull off this move. The effective reward of other stakers is the same between the situation of the lender staking instead and someone borrowing to frontrun a harvest.

Because of these two reasons we believe a High classification is not warranted. We did take up the suggestion to distribute rewards over time. In commit `38d9e69` we modified the `StakingRewards` contract to stream out rewards after a harvest over the week following it.

Low Findings

1. Low - Insufficient signature validation

`Proxy.vy` implements ERC-1271, defining a signature verification method for external contracts.

Technical Details

`Proxy.isValidSignature()` doesn't verify the relation between its parameters `_hash` and `_signature`: the returned value is solely determined by `self.messages[_hash]`. As a result, calls to `Proxy.isValidSignature()` with the same `_hash` but different `_signature` will **all** return the same value, potentially reporting an incorrect validity status to integrating contracts.

Impact

Low. `Proxy` may report an incorrect validity status for a signature.

Recommendation

Also verify `_signature`'s validity against the provided `_hash`.

Developer Response

This is intentional, as we do not define a signature scheme. The act of "signing" a message here is simply changing a flag in the hashmap, the specific signature data does not matter.

2. Low - `GaugeRewards.harvest()` doesn't transfer rewards from the `YearnGauge`

`GaugeRewards.harvest()` can be used only if a call to YearnGauge is done before the execution of the call.

Technical Details

The function `harvest()` is called with two lists: `_gauges` and `_amounts`, corresponding to the gauges to be harvested and the amount of tokens to harvest, respectively. The harvest process then transfers the tokens from the `Gauge.vy` contract to the `GaugeRewards.vy` contract. New tokens are transferred on a `_deposit()` or `_withdraw()` call, but these two calls trigger `report()`, which in turn triggers `_harvest`.

The caller of `harvest()` must call `YearnGauge(asset).getReward(proxy)` before executing the `harvest()` call. The contract's design, involving a for loop over gauges, suggests that the call to `YearnGauge(asset).getReward(proxy)` might be missing to make the harvesting call efficient.

Impact

Low. We want to ensure this is the intended design and the call to collect the `reward_token` isn't missing.

Recommendation

Transfer the reward from the `YearnGauge` to the Gauge on or before `harvest`.

Developer Response

This is intentional. We don't have to enforce calling of the `getReward` function of the yGauge because it is permissionless. And because of this it could theoretically be the case that some rewards have already been sent to the gauge by someone, waiting for a harvest. In that case the harvester has the option to skip the `getReward` call, saving some gas.

3. Low - Prevent unnecessary locking

The contract allows for the relocking of tokens even when it might not be necessary. This behavior can lead to a suboptimal user experience, where users may inadvertently extend the lock duration of their funds, possibly restricting their access to liquidity.

Technical Details

The contract provides a function lock that enables users to lock their staked assets for a specified duration, ostensibly to increase their voting power or to align with specific staking incentives. This function calculates a new lock duration based on the input while considering the existing lock, if any. It's designed to prevent shortening an existing lock but allows for extensions.

One potential issue with this approach is that it does not explicitly prevent unnecessary relocking. Users who interact with the contract may extend the lock duration without a clear benefit, especially if they do despite having full weight (after 8 weeks).

```python
def test_relock(chain, deployer, alice, staking_token, staking):
    # stake can be locked
    staking_token.mint(alice, UNIT, sender=deployer)
    staking_token.approve(staking, UNIT, sender=alice)
    staking.deposit(UNIT, alice, sender=alice)
    assert staking.unlock_times(alice) == 0
    staking.lock(8 * WEEK, sender=alice)
    unlock = staking.unlock_times(alice)
    chain.pending_timestamp += 9 * WEEK
    chain.mine()
    # Now the lock has expired
    assert staking.vote_weight(alice) == UNIT
    staking.lock(8 * WEEK, sender=alice)
    # Lock changed but vote weight is still the same
    assert staking.unlock_times(alice) != unlock
    assert staking.vote_weight(alice) == UNIT
```

Staking.vy#L324

Impact

Low. Suboptimal user experience.

Recommendation

Prevent users with `time - block.timestamp` over `RAMP_LENGTH` from locking assets.

Developer Response

This has been addressed in commit `b5a846d`. A user will now be protected from extending their lock beyond what is needed to get the max vote weight.

4. Low - Prevent lock without any balance

It's possible for a user to inadvertently set up a lock, even without holding any funds. This lock can reduce their ability to use any tokens they later acquire or deposit. Such a lock restricts token movements without offering any advantage in terms of `vote_weight`.

Technical Details

A lock can be established by a user lacking funds, which results in an entry for the lock in `unlock_times`. Should the user later `deposit()` or gain tokens through a transfer, this lock will hinder the ability to withdraw or move these funds. The `time` recorded in `packed_balances` will be updated to reflect the moment of the transfer or deposit. Despite the locking of the funds, the vote_weight remains unaffected, staying at zero.

Impact

Low.

Recommendation

Prevent lock from being created with zero balance.

Developer Response

Fixed in commit `b5a846d`.

5. Low - Transfer exceeding ETH back to sender when above the slippage tolerance

The system is tolerant to a negative slippage of 0.3% but doesn't prevent a user from sending way too much funds to pay for YFI redemption.

Technical Details

```
File: BasicRedeemer.vy
156 |    value -= value * 3 / 1000
157 |    assert value > 0
158 |    assert _eth_amount >= value, "slippage"
159 |    self.yearn_redemption.redeem(_amount, value=value)
```

BasicRedeemer.vy#L156-L159

Impact

Low. User funds sent in excess are captured by the protocol.

Recommendation

Since a 0.3 negative slippage is allowed, any funds above a 0.3% positive slippage should be returned to the `_account`.

Developer Response

Fixed in `afc6d68`.

6. Low - `unstake()` should distribute any claimable assets before proceeding with the creation of a new stream

The unstaking mechanism on the `Staking.vy` contract doesn't distribute funds `withdrawable` before the creation of a new stream.

Technical Details

```
File: Staking.vy
361 |    time, total, claimed = self._unpack(self.packed_streams[msg.sender])
362 |    self.packed_streams[msg.sender] = self._pack(block.timestamp, total - claimed + _assets, 0)
363 |    log Transfer(msg.sender, empty(address), _assets)
```

If funds are pending being withdrawn, they will not be distributed and will be re-streamed from `block.timestamp`.

Staking.vy#L361-L363

Impact

Low.

Recommendation

Transfer withdrawable funds if there are any before recreating a stream.

Developer Response

This has been intentionally left out, to keep the code simple. The user always has the option to withdraw from their existing stream right before starting a new one.

7. Low - Prevent sending ETH to `empty(address)`

`BasicRedeemer.claim_excess()` may be called by anyone to transfer the contract's accrued ETH to `treasury`.

Technical Details

Given that `BasicRedeemer.set_treasury()` doesn't validate the new `treasury` address to be different from `empty(address)`, a mistake in calling the method could result in an external actor being able to transfer the protocol's ETH to `address(0)`.

Impact

Low. Unlikely mistake by the contract's owner may lead to fund loss.

Recommendation

Either prevent setting `self.treasury = empty(address)` within `BasicRedeemer.set_treasury()`, or assert `self.treasury != empty(address)` within `BasicRedeemer.claim_excess()`.

Developer Response

Fixed in `a92e0ff`.

8. Low - Prevent deposits from being credited to a `Gauge`, `LiquidLocker` or `Staking`

Gauge.vy, LiquidLocker.vy and Staking.vy all implement the ERC20 standard, with ad hoc minting logic. These contracts have in common the fact that their `transfer()` and `transferFrom()` methods prohibit users from sending tokens to `empty(address)` or the contract itself.

Technical Details

The contracts listed fail to enforce that deposits must not be credited to the contract itself, making the newly minted tokens unrecoverable.

In particular, `Staking._deposit()` also allows for deposits being credited to `empty(address)`.

PoC

Impact

Low. A user error can cause the contracts to hold funds, which won't be recoverable.

Recommendation

Within each contract's deposit methods, enforce that tokens may not be minted to the contract itself.

- `Gauge._deposit()`:

```
def _deposit(_assets: uint256, _receiver: address):
    """
    @notice
        Handle a deposit by claiming rewards, reporting to the rewards contract
        and transferring tokens from the caller to the proxy
    """
-   assert _assets > 0
+   assert _assets > 0 and _receiver != self
    pending: uint256 = self._pending()
    rewards.report(asset, empty(address), _receiver, _assets, pending)
    assert ERC20(asset).transferFrom(msg.sender, proxy, _assets,
default_return_value=True)
    log Deposit(msg.sender, _receiver, _assets, _assets)
    log Transfer(empty(address), _receiver, _assets)
```

- `LiquidLocker._mint()`:

```
def _mint(_amount: uint256, _receiver: address):
    """
    @notice Mint an amount of liquid locker tokens
    """
-   assert _amount > 0
+   assert _amount > 0 and _receiver != self
    assert _receiver != empty(address)

    self.totalSupply += _amount
    self.balanceOf[_receiver] += _amount
    log Transfer(empty(address), _receiver, _amount)
```

- `Staking._deposit()`:

```
def _deposit(_assets: uint256, _receiver: address):
    """
    @notice Update balance and transfer liquid locker tokens in
    """
+   assert _receiver != self and _receiver != empty(address)
    self.totalSupply += _assets
    self._update_balance(_assets, _receiver, INCREMENT)

    assert ERC20(asset).transferFrom(msg.sender, self, _assets,
default_return_value=True)
    log Deposit(msg.sender, _receiver, _assets, _assets)
    log Transfer(empty(address), _receiver, _assets)
```

Developer Response

Fixed in `c5500a4`.

Gas Saving Findings

1. Gas - Unused function parameter

The `_redeem_yearn()` and `_redeem_curve()` take the `_receiver` but don't use it.

Technical Details

```
File: BasicRedeemer.vy
```

```
151 |  def _redeem_yearn(_receiver: address, _amount: uint256, _eth_amount: uint256):


162 |  def _redeem_curve(_receiver: address, _dt_amount: uint256, _sell_amount: uint256):
```

BasicRedeemer.vy#L151 BasicRedeemer.vy#L162

Impact

Gas savings.

Recommendation

Remove the unused variable.

Developer Response

Now that we refund any positive slippage above 0.3% this parameter is used, so no change is required.

2. Gas - Initialize variables outside the loop

Initialization of a variable inside a loop causes additional unnecessary gas usage.

Technical Details

```
File: GaugeRewards.vy
```

```
127 |   balance: uint256 = 0
128 |   account_integral: uint256 = 0


174 |   gauge: address = _gauges[i]
175 |    amount: uint256 = _amounts[i]
176 |    fees: uint256 = amount * fee_rate / FEE_DENOMINATOR
```

GaugeRewards.vy#L127-L128 GaugeRewards.vy#L174-L176

Proof of concept can be found here.

Impact

Gas savings.

Recommendation

Declare variables outside the loop.

Developer Response

This wouldn't decrease gas cost, as vyper already reuses the same memory for the variable.

3. Gas - Zero address checks are superfluous

Some checks can be removed to improve the contract's gas consumption.

Technical Details

`Staking.vy#L142` `LiquidLocker.vy#L141` `Gauge.vy#L165` `Factory.vy#L66` - this check may be removed as `create_from_blueprint` reverts if `code_offset >= target.codesize`

Impact

Gas savings.

Recommendation

Consider removing the highlighted assertions.

Developer Response

Removed assertion in factory in commit `b624704`.

4. Gas - Math could use unsafe operations

There is a lot of math that could be run with the unsafe math operator to save gas, this reduces the contract readability and might not be necessary in most cases. We have highlighted operations in loops that would benefit more from unsafe math.

Technical Details

```
File: GaugeRewards.vy
131 |    amount += (integral - account_integral) * balance / PRECISION

176 |    fees: uint256 = amount * fee_rate / FEE_DENOMINATOR
177 |    amount -= fees

185 |    total_fees += fees
```

GaugeRewards.vy#L131 GaugeRewards.vy#L176

Impact

Gas savings.

Recommendation

Use unsafe math operations.

Developer Response

We have opted to keep it like it is, prioritizing legibility over the small amount of gas that would be saved.

5. Gas - Do not reduce max allowance

A common gas-saving practice is not to reduce gas allowance when set to `max_value(uint256)`. This removes the need to update a state variable.

Technical Details

```
File: Gauge.vy
349 |         allowance: uint256 = self.allowance[_owner][msg.sender] - _assets
350 |         self.allowance[_owner][msg.sender] = allowance
```

Gauge.vy#L349-L350

```
File: Staking.vy
125 |         allowance: uint256 = self.allowance[_from][msg.sender] - _value
126 |         self.allowance[_from][msg.sender] = allowance

470 |         allowance: uint256 = self.allowance[_owner][msg.sender] - _assets
471 |         self.allowance[_owner][msg.sender] = allowance
```

Staking.vy#L470-L471

```
File: LiquidLocker.vy
126 |         allowance: uint256 = self.allowance[_from][msg.sender] - _value
127 |         self.allowance[_from][msg.sender] = allowance
```

LiquidLocker.vy#L126-L127

Impact

Gas savings.

Recommendation

Do not update the `self.allowance[_owner][msg.sender]` with the new allowance when max allowance is given.

Developer Response

Added in commit `b38db7f`.

6. Gas - Subtractions can be performed on packed data

You can perform the subtraction operation on the packed data before unpacking the values, thereby eliminating the need for additional unpacking and subtraction step.

Technical Details

A demonstration is available here.

Let's examine the following case.

```
File: StakingRewards.vy
    lt_pending: uint256 = 0
    dt_pending: uint256 = 0
    lt_pending, dt_pending = self._unpack(self.packed_pending_rewards[_account])

    lt_integral: uint256 = 0
    dt_integral: uint256 = 0
    lt_integral, dt_integral = self._unpack(self.packed_integrals)
    if _balance == 0:
        # no rewards to be distributed, sync integrals only
        self.packed_account_integrals[_account] = self.packed_integrals
        return lt_pending, dt_pending

    lt_account_integral: uint256 = 0
    dt_account_integral: uint256 = 0
    lt_account_integral, dt_account_integral =
self._unpack(self.packed_account_integrals[_account])
```

Additionally, we'll review the packing function.

```
File: StakingRewards.vy
def _pack(_a: uint256, _b: uint256) -> uint256:
     """
    @notice Pack two values into two equally sized parts of a single slot
    """
    assert _a <= MASK and _b <= MASK
    return _a | _b << 128
```

By packing, the `lt_account_integral` will always be less than or equal to `lt_integral`, ensuring that the `dt_` portion of the integral packed value remains unaltered.

Code that can be changed:

- StakingRewards.vy#L358-L374
- StakingRewards.vy#L123-L134

Impact

Gas savings.

Recommendation

Subtract packed values.

Developer Response

We acknowledge that this would be possible but have opted to not implement this for simplicity.

7. Gas - Unnecessary `assert` check

The `assert` can be moved to the external function, the `_fee_rate` called internally is always under 4.

Technical Details

```
    assert _idx < 4
    return (self.packed_fees >> 32 * (4 + _idx)) & FEE_MASK
```

GaugeRewards.vy#L397

Impact

Gas savings.

Recommendation

Move `assert _idx < 4` to the `fee_rate()` external function.

Developer Response

This has been addressed in commit `df6f33c`.

8. Gas - Unnecessary `assert` on token transfers

In many instances (e.g: 1, 2, 3), the contracts call `transfer()` and `transferFrom()` on YFI and dYFI contracts with `default_return_value == True` and asserting the return data is `== True`.

Technical Details

For these calls, checking the return data to be `True` isn't necessary, as both tokens' implementation will either revert execution or return `True`:

- YFI: `transfer()`, `transferFrom()`
- dYFI: `transfer()`, `transferFrom()`

Impact

Gas savings.

Recommendation

Remove the assertions used in pair with calls to the tokens' `transfer()` and `transferFrom()` methods.

Developer Response

Even though it would not strictly be necessary, we think its good practice to do it everywhere to make sure we don't accidentally forget it somewhere where it would be required.

9. Gas - Avoid balance update on zero-value `transferFrom()` in LiquidLocker.vy

Technical Details

LiquidLocker.vy’s `transferFrom()` checks if the transferred `_value` is greater than 0 before updating the allowance:

```
@external
def transferFrom(_from: address, _to: address, _value: uint256) -> bool:

    // ...

    if _value > 0:
        allowance: uint256 = self.allowance[_from][msg.sender] - _value
        self.allowance[_from][msg.sender] = allowance

    self.balanceOf[_from] -= _value
    self.balanceOf[_to] += _value
    log Transfer(_from, _to, _value)
    return True
```

The same check could be applied before updating the balances.

Impact

Gas. Zero-value transfers will be more expensive than necessary.

Recommendation

Indent the balance updates to be inside the `if _value > 0` branch:

```
@external
def transferFrom(_from: address, _to: address, _value: uint256) -> bool:

    // ...

    if _value > 0:
        allowance: uint256 = self.allowance[_from][msg.sender] - _value
        self.allowance[_from][msg.sender] = allowance
+       self.balanceOf[_from] -= _value
+       self.balanceOf[_to] += _value
-   self.balanceOf[_from] -= _value
-   self.balanceOf[_to] += _value
    log Transfer(_from, _to, _value)
    return True
```

Developer Response

Fixed in `426d5f2`.

10. Gas - Set curve pool as immutable in BasicRedeemer.vy

Since there's already a 50% chance that updating the `curve_pool` requires a redeploy of the redeemer contract, set it as immutable to avoid a storage read on each redemption, and update it by deploying a new contract.

Technical Details

When the `curve_pool.exchange()` function is called on `_redeem_curve()`, the address of the curve pool is loaded from storage, but the coin indexes are hardcoded (see first two arguments):

```
eth_amount: uint256 = self.curve_pool.exchange(0, 1, _sell_amount, 0, True)
```

`set_curve_pool()` allows updating the curve pool. However, if the coin indexes are different in the new pool, a redeploy of the contract is needed.

Impact

Gas. The cost of redemptions that use the curve pool can be substantially reduced.

Recommendation

Set `curve_pool` as immutable, and remove the `set_curve_pool()` function.

> Note: if redeploys are to be avoided, consider storing the coin indexes packed in storage along with the `curve_pool`, and make `set_curve_pool()` receive the new indexes.

Developer Response

Added in commit `426d5f2`

Informational Findings

1. Informational - Unused constant

An unused constant can be removed.

Technical Details

```
File: GaugeRewards.vy
```

```
80 |  HARVEST_FEE_IDX: constant(uint256)     = 0 # harvest
```

[GaugeRewards.vy#L80]

Impact

Informational.

Recommendation

```diff
-     fee_rate: uint256 = (self.packed_fees >> 128) & FEE_MASK
+    fee_rate: uint256 = self._fee_rate(HARVEST_FEE_IDX)
```

Constant should be used on line 169.

Developer Response

This has been fixed.

2. Informational - Typos

Typos.

Technical Details

```
File: contracts/VestingEscrowSimple.vy
67 |     @dev This function is seperate from `__init__` because of the factory pattern
```

```
File: contracts/GaugeRewards.vy
376 |     @notice Harvest a gauge by transfering the reward tokens out of it and updating
the integral
```

VestingEscrowSimple.vy#L67 GaugeRewards.vy#L376

Impact

Informational.

Recommendation

- `seperate` should be `separate`
- `transfering` should be `transferring`

Developer Response

Fixed in `ce8796a`.

3. Informational - `raw_call` should return call data.

A `raw_call()` will not fail in silence with the default value `evert_on_failure = True`, it’s however recommended returning the `value` of the raw call for future usage.

Technical Details

```
File: Proxy.vy
 72 |   raw_call(_target, _data, value=msg.value)
```

Proxy.vy#L72

Impact

Informational.

Recommendation

Return the value returned by `raw_call()`.

Developer Response

We have decided to not return the data as a gas saving measure, as we do not anticipate to need to ready any return data.

4. Informational - Missing event emissions

State changes should be accompanied by an event log, in order to improve with monitoring and analyzing a contract's state through time.

Technical Details

`Staking.lock()` should emit an event when a user locks assets in the contract. `GaugeRewards.claim_fees()` should emit an event upon fee claim. `Proxy.call()` should emit an event when a successful call occurs.

Impact

Informational.

Recommendation

Add suggested event logs.

Developer Response

Added events in commit `075a07b`.

5. Informational - Code is duplicated within the same contract.

Code is duplicated and can be refactored into an internal function.

Technical Details

```
File: Staking.vy
```

```
    old_duration: uint256 = self.unlock_times[msg.sender]
    if old_duration > block.timestamp:
        old_duration -= block.timestamp
    else:
        old_duration = 0


    lock_duration: uint256 = self.unlock_times[_account]
    if lock_duration > block.timestamp:
         lock_duration -= block.timestamp
     else:
         lock_duration = 0
```

```
File: BasicRedeemer.vy
```

```
     assert msg.sender == self.management

     previous: address = self.yearn_redemption.address
     if previous != empty(address):
         # retract previous allowance
         assert discount_token.approve(previous, 0, default_return_value=True)
     if _yearn_redemption != empty(address):
         # set new allowance
         assert discount_token.approve(_yearn_redemption, max_value(uint256),
default_return_value=True)


     assert msg.sender == self.management

     previous: address = self.curve_pool.address
     if previous != empty(address):
         # retract previous allowance
         assert discount_token.approve(previous, 0, default_return_value=True)
     if _curve_pool != empty(address):
         # set new allowance
         assert discount_token.approve(_curve_pool, max_value(uint256),
default_return_value=True)
```

```
File: Staking.vy
104 |    assert _to != empty(address) and _to != self
105 |
106 |    if _value > 0:
107 |        self._update_balance(_value, msg.sender, DECREMENT)
108 |        self._update_balance(_value, _to, INCREMENT)
109 |
110 |    log Transfer(msg.sender, _to, _value)
111 |    return True
```

```
122 |    assert _to != empty(address) and _to != self
123 |
124 |    if _value > 0:
            ...
127 |
128 |        self._update_balance(_value, _from, DECREMENT)
129 |        self._update_balance(_value, _to, INCREMENT)
130 |
131 |    log Transfer(_from, _to, _value)
132 |    return True
```

Staking.vy#L314-L318 Staking.vy#L497-L501 BasicRedeemer.vy#L200-L208 BasicRedeemer.vy#L222-L230 Staking.vy#L104-L111 Staking.vy#L122-132

Impact

Informational.

Recommendation

Use an internal function instead of duplicating the code.

Developer Response

Acknowledged, but we prefer to keep it as is.

6. Informational - Improve code readability

Code readability can be improved by not using the same variable for semantically different values.

Technical Details

- In `Staking.vy#L469-L470`, `claimable` first represents a time span and then an amount of tokens.
- In `GaugeRewards.vy#L143-L145`, `fee` first represents a percentage rate and then an amount of tokens.
- In `StakingRewards.vy#L162-L166`, `lt_fee` first represents a percentage rate and then an amount of tokens.
- In `Staking.vy#L314-L318`, `old_duration` first represents a timestamp and then a time span.

Impact

Informational.

Recommendation

Consider using an extra variable to separate the semantically different variables, or inline the first definition where applicable.

Developer Response

Acknowledged, but we prefer to keep it as is.

7. Informational - Improve events

Values emitted in some events can be changed to provide a more accurate and reliable value for off-chain analysis and monitoring.

Technical Details

- `StakingRewards.Claim` emits a single `fee_idx` value, which refers to the index within the `fee_rates` array used to calculate the dYFI fees charged by the `StakingRewards.claim()` function. This event could benefit by emitting the two fee indexes, `lt_fee_idx` and `dt_fee_idx`, used for YFI and dYFI respectively.

Impact

Informational.

Recommendation

Change events mentioned above.

Developer Response

The fee index for the locking token is implied by the fee index for the discount token, therefore we do not think its worth it to add it to the event.

8. Informational - Check `yearn_redemption != empty(address)` early in `BasicRedeemer.redeem()`

`BasicRedeemer.set_yearn_redemption()` allows for the contract's `management` address to set `yearn_redemption` to an arbitrary value. As per the method's NatSpec documentation, it also allows setting `yearn_redemption = empty(address)` to disable redemptions altogether.

Technical Details

In case dYFI redemption were to be disabled, the highlighted method would execute until hitting L159, in which execution would revert because of a failing `EXTCODESIZE` check.

Impact

Informational.

Recommendation

Within `BasicRedeemer.redeem()`, handle the case in which `self.yearn_redemption.address == empty(address)` by reverting or avoiding to pull dYFI from `msg.sender`.

Developer Response

Since the likelihood of removing the yearn redemption contract is low, we prefer to not do an extra `SLOAD`, optimizing the happy path.

9. Informational - If a yGauge is deregistered there are no checks to pause deposits

If a yGauge is deregistered there are no checks in the system to prevent users to deposit in the corresponding 1up gauge if `management` doesn't deregistered it in time.

Technical Details

When a gauge is created the `deploy_gauge()` function check if the corresponding yGauge is registered.

However, the current implementation lacks mechanisms to verify if a yGauge has been subsequently deregistered. While the `report()` function does perform a check against the 1up registry, this relies on manual updates by the 1up team, leading to potential periods of inconsistency.

Impact

Informational. User who deposit in a deregistered gauge will gains no rewards. As stated here only whitelisted yGauges are able to receive emissions from votes and Governance.

Recommendation

Pause deposits for deregistered yGauges.

```
diff --git a/contracts/Gauge.vy b/contracts/Gauge.vy
index 898f8c0..9bda0f5 100644
--- a/contracts/Gauge.vy
+++ b/contracts/Gauge.vy
@@ -330,9 +330,9 @@ def _deposit(_assets: uint256, _receiver: address):
         Handle a deposit by claiming rewards, reporting to the rewards contract
         and transferring tokens from the caller to the proxy
     """
-    assert _assets > 0
+    assert _assets > 0 and yearn_registry.registered(asset)
     pending: uint256 = self._pending()
     rewards.report(asset, empty(address), _receiver, _assets, pending)
     assert ERC20(asset).transferFrom(msg.sender, proxy, _assets,
default_return_value=True)
     log Deposit(msg.sender, _receiver, _assets, _assets)
     log Transfer(empty(address), _receiver, _assets)
```

Developer Response

We acknowledge this is the case, but prefer to keep it as is to keep the contracts modular. In the theoretical situation where Yearn decides to move to a different registry, it would be easy for us to start using it by swapping out our factory. This would be very hard to do if all gauges had a reference to the yearn registry too.

Final remarks

1upYFI enables users to lock YFI into veYFI and deposit their yGauge tokens, pool their voting power together and benefit from socializing their boost. The protocol accumulates YFI tokens, locks them perpetually for veYFI and mints a liquid representation of its veYFI holding, upYFI. Furthermore, upYFI holders are able to stake their tokens, in order to be eligible to receive further rewards, derived from Yearn's reward stream to veYFI holders. Given that Yearn distributes these rewards in the form of dYFI, 1upYFI offers its users the possibility of redeeming the rewards in 3 ways:

1. Receive dYFI directly.
2. Redeem its dYFI for YFI, by employing the user's ETH.
3. Sell a portion of the user's dYFI for ETH on a Curve pool, in order to use the obtained ETH to redeem the remaining dYFI for YFI.

Future versions may allow for more complex redeem flows, by using another implementation of the `BasicRedeemer.vy` contract.

Users should be aware that the assets collected by the protocol, the veYFI lock and yGauge tokens, are all held by the `Proxy.sol` contract, which defines a set of operators allowed to make any call on the contract's behalf, posing as a centralized point of failure. Finally, the system is designed to be immutable, with some parameters and contract addresses that the owner is able to change arbitrarily.