yAcademy LevGeist review

Review Resources:
Documentation was provided

Residents:

- Peep

Fellows:

- Koby Hall
- datapunk
- pashov
- Jib

Table of Contents

tags: `Review` `yAcademy`

Review Summary

Yearn LevGeist

The levgeist branch of the Yearn-contracts Repo was reviewed over 11 days. 2 days were used to create a report of the findings. The contracts were reviewed from 6/13 to 6/24. The repository was under active development during the review, but the review was limited to commit 6c43c39c2e9cfcfb329edf3678843c5b57095dcf.

Scope

Code Repo

The review covered contracts modified by the pull request at above specific commit and focused on the contracts directory.

After the findings were presented to the Yearn team, fixes were made and included up to commit 6c43c39c2e9cfcfb329edf3678843c5b57095dcf. Fixes were reviewed to confirm whether they remedied the original finding, but a new security review was not performed on the revised code (e.g. to determine whether new vulnerabilities were introduced by the fixes).

The 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.

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 party users that the code has been audited nor that the code is free from defects. By deploying or using the code, yearn and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Access controls are applied where needed. Ownable is appropriately used to limit function calls to respective permissions, and some view functions are made internal accordingly.
Mathematics	Good	Solidity 0.6.12 is used, which provides no overflow and underflow protect, but BaseStrategy already has SafeERC and SafeMath being used. No low-level bitwise operations are performed. There was no unusually complex math.
Complexity	Medium	With the exception of a few no-ops and superfluous logic, nothing was extremely complex or out of the ordinary.
Libraries	Fair	SafeERC20 and Safemath are used as basic libraries for value flow; Yearn BaseStrategy is being used with StrategyLib. Besides for that, simple interfaces are used.
Decentralization	Fair	Currently, Yearn access controls are applied.
Code stability	Average	Changes were reviewed at a specific commit and the scope was not expanded after the review was started.
Documentation	Moderate	Comments existed in little places in Strategy.sol, and auditors were expected to mostly have context on what code is supposed to do. Strategy contracts lacked detailed comments. Some documentation had to be added to distinguish between functions with duplicate names, as well as fixing some typos.
Monitoring	Good	Events were added to all important functions that modified state variables.
Testing and verification	Fair	Test coverage was expansive, but some tests were not passing due to current market conditions.

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 - `Strategy.sol` no slippage tolerance limit set on `_sellRewardForWant` (Pashov, datapunk, Jib)

Proof of concept

`_claimAndSellRewards` always passes 0 as the `minOut` argument to the `_sellRewardForWant` method, so there is no slippage tolerance limit.

Impact

Calling `_claimAndSellRewards` method can be sandwich attacked since there is no slippage tolerance limit and the transaction won't revert even if there was 99% slippage when doing `_sellRewardForWant`.

Recommendation

Add a non-zero value for the `minOut` argument so there is some slippage tolerance limit. If it is too high it can lead to Denial of Service attacks but zero slippage tolerance limit is not much better.

Response

We have not found a good way to provide a minOut value on chain. As such, yearn uses private relays where available. Future versions of this strategy could use ySwaps, which sells asynchronously and can populate minOut off chain.

Low Findings

1. Low - `BaseStrategy.sol` Missing non-zero address check in `setHealthCheck` (Pashov)

Proof of concept

The function does not have a non-zero address check for its address argument. All other setter methods for address type variables in the contract have non-zero address checks.

Impact

Passing the zero address as an argument to the function can lead to wasted gas and some confusion when executing transactions after.

Recommendation

Keep the consistency with other methods in the contract and add a `require(_healthCheck != address(0));` check

2. Low - `liquidatePosition` loss recording does not record loss when `_amountNeeded.sub(_liquidatedAmount) <= minWant`

Proof of concept

Losses are only recorded when `_amountNeeded.sub(_liquidatedAmount); <= minWant`

```
if (diff <= minWant) {
    _loss = diff;
}
```

Instead of `_amountNeeded.sub(_liquidatedAmount) >= minWant`

Impact

If `_amountNeeded.sub(_liquidatedAmount) >= minWant`, then loss will not be recorded.

Recommendation

Use `if (diff >= minWant)` instead.

Response

if (diff <= minWant) is intentional. The goal of this statement is to prevent the vault from moving to the next strategy in the withdraw queue when the diff is “dust-sized”. liquidatePosition does not consider the diff being non-zero to be a loss as it may be temporary illiquity due to debt market utilization.

3. Low - `Strategy.sol` Missing non-zero address check in `_initializeThis` (Pashov)

Proof of concept

The function calls an external contract to get some token addresses

```
(address _aToken, , address _debtToken) = protocolDataProvider.getReserveTokensAddresses(address(want));
```

and then has a non-zero address check for `_aToken`, but not for `_debtToken`.

Impact

If `_debtToken` is set to zero the `_initializeThis` function would have to be called again or, more likely, the contract would have to be redeployed. This will waste gas and will slow down strategy start time.

Recommendation

Keep the consistency with the other received address value and add a `require(_debtToken != address(0));` check

Response

A zero value is allowed so that the healthcheck can be permanently disabled.

4. Low - `Strategy.sol` Missing invariants checks in `liquidatePosition` (Pashov)

Proof of concept

In the code we see the following comments:

```
// NOTE: Maintain invariant `want.balanceOf(this) >= _liquidatedAmount`
// NOTE: Maintain invariant `_liquidatedAmount + _loss <= _amountNeeded
```

But they are just assumptions made by the developer and there are no assert/require statements to ensure they are really correct.

Impact

Since those are very important invariants, if this math doesn't hold, the contract can lose user's funds. Still this hasn't been proved with an actual PoC, hence the Low risk.

Recommendation

Add assert statements at the end of `liquidatePosition` that implement the invariants in the comments.

Response

Acknowledged

Gas Savings Findings

1. Gas - `Strategy.sol` unreachable code in prepareReturn (JIB, datapunk)

Proof of concept

On lines 254 to 257, the if statement:

```
if (amountRequired.sub(_debtPayment) < _profit)
```

does not evaluate, this is because amountRequired is `debtOutstanding + _profit` and `_debtPayment` is `debtOutstanding`, meaning that the if statement always evaluates to `_profit < _profit` which is never true, meaning that line 256 never evaluates.

Impact

The omission of this line seemingly has no impact and so is unnecessary code.

Recommendation

Delete lines 254 to 257. This issue is also present in LevAave between lines 307 and 310 and lines 329 and 331.

2. Gas - Unnecessary liquidation procedures (Jib)

Proof of concept

On line 336 https://github.com/Yacademy-block-2/yearnV2-gen-lev-lending/blob/levgeist/contracts/Strategy.sol#L336 the function checks if the want balance is sufficient to cover the amountNeeded, this is set as `wantBalance > _amountNeeded`. If wantBalance is the same as _amountNeeded the strategy still has sufficient funds to cover the withdraw, meaning the logic that follows this if statement is unnecessary.

Impact

The impact is low, it is a gas saving to prevent unnecessary logic execution

Recommendation

Change to `wantBalance >= _amountNeeded`

Response

Acknowledged

3. Gas - `Strategy.sol` & `BaseStrategy.sol` inefficient usage of for loops (Pashov)

Proof of concept

`Strategy.sol` lines 199, 455, 509 and `BaseStrategy.sol` line 893 contain for loops that are implemented inefficiently in terms of gas for the EVM. Omitting assigning a default-zero type variable to zero, caching array's length, using `++i` instead of `i++` can save a good chunk of gas, especially if the loop is long running.

Impact

Gas savings for protocol interactors

Recommendation

1. Always use uint256 as the type for the index counter, because of EVM's 256 bit word size. Current implementation with uint8 uses more gas than if it used uint256.
2. Don't initialise index counter variable to zero. Zero is its default value and reinitialising to zero costs extra gas
3. When the for loop end check expression is checking for some array's length (`Strategy.sol` line 199), always cache the length of the array in a separate stack variable.
4. Use `++i` instead of `i++` in for loops (small gas saving)

5. You can wrap the `++i` in an unchecked block since you have a for loop end check expression anyway

Example for loop implemented using the recommendations:
Current (`Strategy.sol` line 199)

```
for (uint8 i = 0; i < rewards.length; i++) {
    rewardBalance += rewards[i];
}
```

With recommended gas optimisations:

```
uint256 rewardsLength = rewards.length;
for (uint256 i; i < rewardsLength;) {
    rewardBalance += rewards[i];

    unchecked {
        ++i;
    }
}
```

4. Gas - Regular math can be used instead of SafeMath when it is safe (datapunk)

Proof of concept

```
    if (totalDebt > totalAssets) {
        // we have losses
        _loss = totalDebt.sub(totalAssets);
    } else {
        // we have profit
        _profit = totalAssets.sub(totalDebt);
    }
```

Impact

Save some gas

Recommendation

Use regular math instead

```
    if (totalDebt > totalAssets) {
        // we have losses
        _loss = totalDebt - totalAssets;
    } else {
        // we have profit
        _profit = totalAssets - totalDebt;
    }
```

Response

Acknowledged

Informational Findings

1. Informational - `BaseStrategy.sol` does not emit event in `setHealthCheck` (Pashov)

Proof of concept

All other setter methods for address type variables in the contract emit events on state change, example:

```
function setStrategist(address _strategist) external onlyAuthorized {
        require(_strategist != address(0));
        strategist = _strategist;
        emit UpdatedStrategist(_strategist);
    }
```

Impact

Not following the contract convention might lead to some confusion and unexpected behaviour. Also there can't be any code that can be subscribed to healthCheck address change events.

Recommendation

Follow the contract convention and emit a state change event like

```
emit UpdatedHealthCheck(_healthCheck);
```

Response

Acknowledged

2. Informational - `Strategy.sol` does not emit event in `setCollateralTargets`, `setMinsAndMaxs`, `setRewardBehavior` (datapunk)

Impact

Not following the contract convention might lead to some confusion and unexpected behaviour. Also there can't be any code that can be subscribed to important change events.

Recommendation

Follow the contract convention and emit a state change event like

```
emit SetCollateralTargets(_targetCollatRatio, _maxCollatRatio, _maxBorrowCollatRatio);
emit SetMinsAndMaxs(minWant, _minRatio, _maxIterations);
emit SetRewardBehavior(_swapRouter, _minRewardToSell);
```

3. Informational - `Strategy.sol` has rewards local variable defined in a couple of places which shadows reward state variable as defined in `BaseStrategy.sol` (datapunk)

Impact

Variable shadowing might lead to some confusion and unexpected behaviour.
https://github.com/Yacademy-block-2/yearnV2-gen-lev-lending/blob/6c43c39c2e9cfcfb329edf3678843c5b57095dcf/contracts/Strategy.sol#L187
https://github.com/Yacademy-block-2/yearnV2-gen-lev-lending/blob/6c43c39c2e9cfcfb329edf3678843c5b57095dcf/contracts/Strategy.sol#L197

Recommendation

Rename the rewards local variables to something else.

Response

Acknowledged

4. Informational - Strategy.sol uses `now` keyword in `_sellRewardForWant` (Pashov)

Proof of concept

The code makes use of the `now` keyword in Solidity which is deprecated in future Solidity versions.

Impact

If contract gets upgraded to a newer solidity version it won't compile.

Response

Acknowledged

Recommendation

Use `block.timestamp` instead of `now`

5. Informational - Strategy.sol `estimatedTotalAssets` can be declared as `external` (Pashov)

Proof of concept

The `estimatedTotalAssets` method is declared as `public` which should be used for method that will be called from inside of the contract, but `estimatedTotalAssets` isn't.

Impact

No impact, just following best practices and saving a few seconds for the person reading the code.

Recommendation

Declare `estimatedTotalAssets` as `external` instead of `public`

Response

Acknowledged

6. Informational - Centralised control of LP tokens on Spookyswap (Jib)

Proof of concept

As of 24/06/2022 84.6% of all Geist/FTM LP tokens are controlled by a single address as seen here: https://ftmscan.com/token/0x668ae94d0870230ac007a01b471d02b2c94ddcb9#balances this address is a MasterChef so does decentralise the system slightly, however if for whatever reason this liquidity was pulled it could result in extremely high slippage which reduces profitability of the strategy and exacerbates the chances of a sandwich attack.

Impact

Since the large holder is a master chef contract which will have multiple addresses depositing to it, the risk is somewhat mitigated. However, should the master chef stop distributing rewards or cease from working for some other reason then an issue could emerge.

Recommendation

Monitor the LP profile of the GEIST/FTM pair and if liquidity drops too far then it is unlikely the strategy is going to be overly profitable.

Response

Acknowledged

Appendix and FAQ

tags: `Review` `yAcademy`