yAudit Yearn BalancerLpFactory Review

Review Resources:

- Additional documentation was not provided, but a presentation was given regarding contract functionalities.

Auditors:

- Peep (xiangan.eth)

All fellows participated on this audit. Fellows' names are listed on each finding.

Table of Contents

1 Review Summary
2 Scope
3 Code Evaluation Matrix
4 Findings Explanation
5 High Findings
   a 1. High - Dependency on a single DEX for CRV price (blockdev, pashov)
      a Proof of concept
      b Impact
      c Recommendation
      d Developer Response
   b 2. High - Possible revert on creating a vault (blockdev, devtooligan, SaharAP)
      a Proof of concept
      b Impact
      c Recommendation
      d Developer Response
   c 3. High - Strategy migrations fail to claim reward/extraReward tokens (Benjamin Samuels)
      a Impact

- b Recommendation
- c Developer Response

d 4. `_loss` on `liquidatePosition()` is never accounted in strategy & vault, which can result in funds lost/stuck (pashov, Jib)

- a Proof of concept
- b Impact
- c Recommendation
- d Developer Response

e 5. Dependence on Curve/Balancer governance (devtooligan, Jib)

- a Impact
- b Recommendation
- c Developer Response

f 6. Wrong parameter to cloneStrategyConvex function (datapunk)

- a Proof of concept
- b Impact
- c Recommendation
- d Developer Response

6 Medium Findings

a 1. Medium - Strategy migrations will cause inaccurate accounting on first harvest of new strategy (Benjamin Samuels)

- a Impact
- b Recommendation
- c Developer Response

b 2. Convex's extraRewards array is unbounded (pashov)

- a Proof of concept
- b Impact
- c Recommendation
- d Developer Response

7 Low Findings

a 1. Low - _loss incorrectly assumed even if rewards can be sold to cover loss (Jib)
  a Impact
  b Recommendation
  c Developer Response

8 Gas Savings Findings
a 1. Gas - Setting deposit limit twice (blockdev, devtooligan, datapunk)
  a Proof of concept
  b Impact
  c Recommendation
  d Dev Reply
b 2. Gas - External call on each loop iteration (blockdev)
  a Proof of concept
  b Developer Response
  c Impact
  d Recommendation
c 3. Gas - Use of SafeMath for safe arithmetic (blockdev)
  a Proof of concept
  b Impact
  c Recommendation
  d Developer Response
d 4. Gas - Remove unused state variable (SaharAP)
  a Proof of concept
  b Impact
  c Recommendation
  d Developer Response
e 5. Gas - Use >0 for unsigned integers (SaharAP)
  a Proof of concept
  b Impact
  c Recommendation

- d Developer Response
- f 6. Storage variable `numVaults` in BalancerGlobal is not needed (pashov)
  - a Proof of concept
  - b Recommendation
  - c Developer Response
- g 7. recalculation not needed (datapunk)
  - a Proof of concept
  - b Recommendation
- h 8. remove `tradesEnabled` (datapunk)
  - a Proof of concept
  - b Recommendation
  - c Developer Response
- i 9. Iterating through `rewardsTokens` in `_setUpTradeFactory` and `_removeTradeFactoryPermissions` (verypoor)
  - a Proof of concept
  - b Recommendation
- j 10. Redundant external call when using `staticcall` (verypoor)
  - a Proof of concept
  - b Recommendation

9 Informational Findings

- a 1. Upgrade Pragma (devtooligan)
  - a Impact
  - b Recommendation
  - c Developer Response
- b 2. Informational - Missing events in BalancerGlobal & StrategyConvexFactoryClonable (Benjamin Samuels, pashov, datapunk)
  - a Developer Response
- c 3. ERC20.safeApprove is deprecated (pashov)
  - a Proof of concept

- b Impact
- c Recommendation
- d Developer Response

d 4. Informational- incorrect comment (Jib)

- a Proof of concept
- b Impact
- c Recommendation
- d Developer Response

e 5. Informational- hardcoded addresses (verypoor)

- a Proof of concept
- b Impact
- c Recommendation
- d Developer Response

10 Out of Scope findings

a 1. Medium - `VaultRegistry.latestVault()` functions can consume all gas (blockdev)

- a Proof of concept
- b Impact
- c Recommendation

11 Final remarks

a Peep

12 About yAcademy

Review Summary

Yearn BalancerLpFactory

BalancerLpFactory is a CRV / CVX farming strategy deployed on Balancer.

The master branch of the BalancerLpFactory Repo was reviewed over 13 days. The contracts were reviewed by 1 auditor and several Fellows from June 29 to July 12, 2022. 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 a718dc56465bf1f0881389ad1801ac9447aee47d. The review covered contracts modified by the pull request at this specific commit and focused on the contracts directory.

After the findings were presented to the Yearn dev, fixes were made and included up to commit a718dc56465bf1f0881389ad1801ac9447aee47d.

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.

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 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	Complexity seems relatively similar to other Yearn strategies reviewed: no extraordinarily complex functions

Category	Mark	Description
		or logic, nor no-ops located in code.
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.

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 - Dependency on a single DEX for CRV price (blockdev, pashov)

Relying on a single DEX for an asset price is not ideal as it may not reflect the asset's market price. Here, price of CRV is calculated in terms of USDT using only Sushiswap.

Proof of concept

StrategyConvexFactoryClonable.sol#L430 estimates value of CRV tokens in terms of USDT by routing it through WETH swaps.

Impact

High. Estimating correct price of CRV in terms of USD is important as it impacts the decision of harvesting a strategy. If any of CRV-WETH or WETH-USDT price deviates from the market price, harvest is impacted.

Recommendation

Yearn clarified that they need to estimate CRV price in USD, so dependence on USDT is not needed. In this case, Chainlink oracle should be used to estimate CRV's price in USD. To add more robustness, a second on-chain TWAP oracle can also be used.

Developer Response

Complete.

2. High - Possible revert on creating a vault (blockdev, devtooligan, SaharAP)

Creating a vault and a strategy will fail if there is the vault registry has no vault registered for LP token for `DEFAULT` and `AUTOMATED` type.

Proof of concept

BalancerGlobal.sol#L368-L375:

```
bytes memory data =
    abi.encodeWithSignature("latestVault(address)", lptoken);
(bool success, ) = address(registry).staticcall(data);
if (success) {
    return registry.latestVault(lptoken);
```

```
} else {
    return registry.latestVault(lptoken, VaultType.AUTOMATED);
}
```

If no vault is registered for DEFAULT, `success` is `false`, and since there is no AUTOMATED type vault, the else clause revert. So in this case, it becomes impossible to create a vault for `lptoken`.

Impact

High. It impacts the protocol decentralization, as for each new `lptoken`, `BalancerGlobal`'s `owner` or `management` has to deploy a vault which can cause a delay in operations.

Recommendation

If the call to `latestVault(lptoken, VaultType.AUTOMATED)` fails, return zero address as shown in this diff :

```
  if (success) {
      return registry.latestVault(lptoken);
  } else {
+     data = abi.encodeWithSignature("latestVault(address, VaultType)", lptoken,
VaultType.AUTOMATED);
+     (success, ) = address(registry).staticcall(data);
+ }
+
+ if (success) {
      return registry.latestVault(lptoken, VaultType.AUTOMATED);
+ } else {
+     return address(0);
  }
```

Developer Response

Good catch. A change has been implemented in the registry code to return 0x0 instead of reverts. This allows the factory logic to be greatly simplified. An update has been pushed.

3. High - Strategy migrations fail to claim reward/extraReward tokens (Benjamin Samuels)

When StrategyConvexFactoryClonable is migrated, the `prepareMigration(...)` function fails to claim & sell extra reward tokens (CRV, CVX, etc.).

Impact

There does not appear to be any mechanism to harvest or re-enable a strategy that has been migrated. This means that any unharvested rewards are effectively lost on migration.

The magnitude of this finding depends on the amount of time that has passed since the strategy was last harvested. If the strategy has not been harvested in a week & is migrated, then a week's worth of extra rewards are lost.

Given the following assumptions/observations, this finding may have a high impact:

1. Some Curve vaults go unharvested for many weeks at a time.
2. CRV/CVX/Extra rewards tend to make up a large fraction of yield revenue.
3. There is no mechanism to prevent a strategy migration when a strategy has not been harvested in a long period of time.

If the magnitude/accuracy of the above assumptions are incorrect, then this finding may have medium or lower severity.

Recommendation

Require harvests to occur before migrating strategies to new versions. It appears that normal harvests call into `prepareReturn(...)`, which claims & sells extra reward tokens. Whether a pre-migration harvest should be implemented in code or as an operational procedure is up to the team.

Developer Response

Good feedback. This was an intentional decision due to the fact that some rewards tokens can potentially cause revert on migration. PrepareMigration is an extremely sensitive function as a revert on migration could lead to loss of capital.

As middleground, we have decided to transfer CRV and CVX tokens only. And will allow additional rewards tokens, if relevant, to be swept and transferred by governance.

4. `_loss` on `liquidatePosition()` is never accounted in strategy & vault, which can result in funds lost/stuck (pashov, Jib)

Proof of concept

In `StrategyConvexFactoryClonable.sol` line 348 we have the following code `(uint256 freed, ) = liquidatePosition(toFree);` which ignores second return value, which is actually the `_loss` from liquidating a position. As a result the `_loss` return value from `prepareReturn()` that `BaseStrategy.sol` uses for accounting/reporting and healthcheck will be with a default value of 0 even though there was a loss.

Impact

In `BaseStrategy.sol`, in `harvest()` we have this code `debtOutstanding = vault.report(profit, loss, debtPayment);` which reports the loss to the vault, so if it reports a 0 loss when there is one then the whole accounting logic for the strategy will be incorrect which can lead to loss funds or stuck funds in strategy. Also the correctness of the healthcheck in the strategy will not be certain because of the following code in `BaseStrategy.sol` that uses `loss`: `require(HealthCheck(healthCheck).check(profit, loss, debtPayment, debtOutstanding, totalDebt), "!healthcheck");`

Recommendation

Change `(uint256 freed, ) = liquidatePosition(toFree);` to `(uint256 freed, _loss) = liquidatePosition(toFree);` - this will directly set the `_loss` return value of `prepareReturn`

Developer Response

Referenced Line: https://github.com/flashfish0x/BalancerLpFactory/blob/a718dc56465bf1f0881389ad1801ac9447aee47d/contracts/StrategyConvexFactoryClonable.sol#L345

The code, as it stands, will both detect and properly account for any loss of funds from the gauge or Aura voter itself (when comparing `assets` to `debt`).

What you point out here only covers the edge case of Aura not honoring a 1:1 redemption of LPs when calling `withdrawAndUnwrap`. While acknowledged, we think this is extremely unlikely edgecase and do not think adding complexity to the accounting is a worthy tradeoff. If any issues arise, strategies can be migrated to a fixed version.

5. Dependence on Curve/Balancer governance (devtooligan, Jib)

Currently, Yearn is wholly dependent on the Curve/Balancer governance approval process for pools that have gauges added to the gauge controller. It has been noted that this has worked quite well until now, with only 1 “ruggy” situation (USDM) having come up.

Impact

Things move quick in this space and “Curve wars” bribing adds additional volatility to the situation. With this new automated process for adding pools/strategies, the risk from dependence on external governance processes is amplified. In the event of another governance attack or if a vulnerable token were to enter the system, then malicious actions could be completed more quickly now due to this new permissionless system dependent on approved gauges.

Recommendation

Review existing capabilities to shut down or pause parts of the protocol if a situation were to arise. If necessary, add new functionality to pause certain functionality of vaults and strategies for a given gauge for example preventing additional inflows to the pool.

Consider whitelisting pools as they are approved for addition to the gauge controller. A small mitigation would be to implement a timelock or other mechanism so that there was a period of time between the guage addition and it’s acceptance by Yearn. Ideally, Yearn would have their own governance process for approving the new pools that get added to the whitelist. Things change quickly and, even though the Curve/Balancer governance is working today, that dependency is not a long term solution. In the words of charlie_eth from the Curve team, “Permisionless pool factories and permisionless gauges are meant to empower governance which comes with serious responsibilities.”

Developer Response

I would submit that the vision here is for these factory compounder vaults to be viewed more as a permissionless utility than an actively managed product. Thus, Yearn should take an optimistic approach, err’ing on the side of rapid deployment and low-friction. While also reserving the right of governance some recourse to set `vault.depositLimit()` to 0 (effectively blocking new deposits).

Further, a user’s decision to LP in a risky pool is not Yearn’s concern. As Yearn LP vaults simply accept LP tokens and can do nothing to convert them back to user’s desired token.

6. Wrong parameter to cloneStrategyConvex function (datapunk)

Proof of concept

The 3rd parameter in cloneStrategyConvex() is a `_rewards` address as defined in BalancerGlobal.sol#L83

```
interface IStrategy {
    function cloneStrategyConvex(
        address _vault,
        address _strategist,
        address _rewards,
        address _keeper,
        uint256 _pid,
        address _tradeFactory,
        uint256 _harvestProfitMax,
        address _booster,
        address _convexToken

    ) external returns (address newStrategy);
```

However, the same management address was passed in for both 2nd and 3rd parameters: BalancerGlobal.sol#L479

Impact

Rewards tokens will be incorrectly attributed to a management address intread of treasury address

Recommendation

Change it to

```
strategy = IStrategy(auraStratImplementation)
    .cloneStrategyConvex(
    vault,
    management,
-   management,
+   treasury,
    keeper,
    pid,
    tradeFactory,
    harvestProfitMaxInUsdt,
    address(booster),
```

```
    aura
);
```

Developer Response

Great catch. Fixed.

Medium Findings

1. Medium - Strategy migrations will cause inaccurate accounting on first harvest of new strategy (Benjamin Samuels)

When StrategyConvexFactoryClonable is migrated, the `prepareMigration` function withdraws tokens from Convex without accounting for any gain or loss since the last harvest.

The balance of want tokens is then transferred to the new strategy as unassigned debt, and the new strategy is assigned the same debtOutstanding as the old strategy.

The first time the new strategy is harvested, it will realize the gain/loss of the old strategy's withdrawl.

Impact

The impact of this finding is dependent on Yearn's strategy managment practices & downstream tooling, so consider this impact analysis "best effort".

If the old strategy has not been harvested in a long time, there may be a relatively large amount of unreported gains/losses that have yet to be realized by the old strategy.

This may cause data accuracy problems when analyzing the APR performance for the new strategy. Consider the case where:

1. Old strategy was last harvested a week ago.
2. Old strategy is migrated to new strategy.
3. 10 minutes later, the new strategy is harvested.

In the above scenario, a week's worth of gains/losses will be realized by the new strategy, and since the new strategy was deployed 10 minutes prior, off-chain tooling/instrumentation might draw inaccurate conclusions about the performance of the

new strategy.

Recommendation

Trigger a harvest before migrating to a new strategy version.

If this is implemented as an operational requirement for migration (rather than coding it), that procedure should be documented on the migration function.

Developer Response

This is a known issue in all strategy migrations. Typically a harvest + migration can be done in same multisig transaction to resolve it - but depends on governance remembering to do so in the proper sequence. We consider this minimal impact and will choose to take no action here.

2. Convex's extraRewards array is unbounded (pashov)

Proof of concept

StrategyConvexFactoryClonable#_updateRewards has the following for loop:

```
        for (uint256 i; i < rewardsContract.extraRewardsLength(); i++) {
            address virtualRewardsPool = rewardsContract.extraRewards(i);
            address _rewardsToken =
                IConvexRewards(virtualRewardsPool).rewardToken();

            // we only need to approve the new token and turn on rewards if the extra
rewards isn't CVX
            if (_rewardsToken != address(convexToken)) {
                rewardsTokens.push(_rewardsToken);
            }
        }
```

This basically loops over Convex's extra rewards. The problem is that adding extra rewards is not bounded in Convex link. This means that if there are too many extra rewards this function will run out of gas/go over the block gas limit and result in a DoS of core strategy functionality (updating rewards).

Impact

When attack is executed (there are too many extra rewards added in Convex) the strategy can lose its option to call `updateRewards`.

Recommendation

Add an `offset` param to `updateRewards`, which you can use to offset the array's index that you use to call `rewardsContract.extraRewards()` with. Such an offset is implemented here

Developer Response

Marking this as low priority. We have `turnOffRewards` and `sweep` which allows us to bypass any critical issues.

Low Findings

1. Low - _loss incorrectly assumed even if rewards can be sold to cover loss (Jib)

When the strategy calls `liquidatePosition` on L564 it unwraps enough funds to cover the `_amountNeeded`. If the amount liquidated is not large enough to cover the `_amountNeeded` then on L571 a `_loss` is recorded with `_amountNeeded.sub(_liquidatedAmount)`. However, there may still be rewards that could be sold to cover the remaining amount, meaning the strategy has incorrectly suggested it has taken a loss.

Impact

In `BaseStrategy.sol`, in `withdraw()` the `_loss` will be returned to the vault. The vault will then report this loss, suggesting the price per share is lower, despite the fact that the strategy could have even been in a profit if rewards were sold.

Recommendation

`liquidatePosition` is only used in 2 places, in `withdraw` and in `prepareReturn`. In `prepareReturn` the `_loss` value isnt used. In withdraw the `_loss` value is read. In normal operation this ideally should not return a `_loss` (so removing L571 could be a fix). However, there could be a blackswan scenario where the strategy has experienced an exploit, meaning the loss is correct, mis-recording a loss in this scenario would mean that depositors who are aware of the exploit could get away with more funds than allowed. This decision is left to the developer. (It has been put to low as in my opinion, as it currently is, is likely the better way).

Developer Response

Agree with your assessment. Choice is to keep as is.

Gas Savings Findings

1. Gas - Setting deposit limit twice (blockdev, devtooligan, datapunk)

Same operation of setting deposit limit on a vault is done twice.

Proof of concept

BalancerGlobal.sol#L463-L472

`v.setDepositLimit(depositLimit)` and `Vault(vault).setDepositLimit(depositLimit)` both set the deposit limit to the same value. The second call is not changing the deposit limit.

Impact

Gas savings.

Recommendation

Remove the second call. This has been fixed in a later commit.

Dev Reply

This was from an older commit and was already fixed.

2. Gas - External call on each loop iteration (blockdev)

If iterating over an array, the loop can be gas-optimized by storing the array length in memory. It saves gas since at each iteration, the loop variable is checked against the length.

Proof of concept

StrategyConvexFactoryClonable.sol#L484

```
for (uint256 i; i < rewardsContract.extraRewardsLength(); i++) {
```

Here at each iteration an external static call is made to fetch the array length.

Developer Response

Fixed.

Impact

Gas savings

Recommendation

Store the length in memory and replace it with the external call. Replace the code above with:

```
uint256 length = rewardsContract.extraRewardsLength();
for (uint256 i; i < length; i++) {
```

3. Gas - Use of SafeMath for safe arithmetic (blockdev)

If some operation is guaranteed to not overflow or underflow, use of OpenZeppelin's SafeMath library can be avoided to save gas.

Proof of concept

StrategyConvexFactoryClonable.sol#L343:

```
_profit = assets.sub(debt);
```

StrategyConvexFactoryClonable.sol#L361:

```
_loss = debt.sub(assets);
```

StrategyConvexFactoryClonable.sol#L565:

```
Math.min(_stakedBal, _amountNeeded.sub(_wantBal)),
```

In all these case, SafeMath can be avoided.

Impact

Gas savings

Recommendation

Replace SafeMath `sub()` with the vanilla subtraction operator: `-`.

Developer Response

Good suggestion. Ignoring for now.

4. Gas - Remove unused state variable (SaharAP)

Proof of concept

state variable `numVaults` in `BalancerGlobal` contract has been set once in createNewVaultsAndStrategies and have never been used any other function. There is an additional `SSTORE` operation each time we want to create a new vault.

Impact

Gas Savings

Recommendation

`numVaults` should be removed and if the variable is useful for external use, it can be defined in a `view` function which returns `deployedVaults.length`.

Developer Response

This has been resolved via another finding #6 below which made the suggestion to remove the state variable and use a getter returning array length instead.

5. Gas - Use >0 for unsigned integers (SaharAP)

Proof of concept

!= 0 is a cheaper operation compared to >0, when dealing with uint. >0 can be replaced with != 0 for gas optimization. The >0 has been used in many places in `BalancerGlobal` and `StrategyConvexFactoryClonable` contract such as here and here.

Impact

Gas Savings

Recommendation

Replace >0 with !=0 when comparing unsigned integer variables to save gas.

Developer Response

Ignored as low.

6. Storage variable `numVaults` in BalancerGlobal is not needed (pashov)

Proof of concept

The variable is set only once in `numVaults = deployedVaults.length;`. There is no need to use a separate storage slot for this value, you can just add a getter method for

`deployedVaults.length` instead

Recommendation

Add a getter method for `deployedVaults.length` and remove `numVaults` storage variable.

Developer Response

Great. Fix has been taken.

7. recalculation not needed (datapunk)

Proof of concept

As marked in the the snippet below, there is no need to recalculate `_profit.add(_debtPayment)` and `_profit` In L#345

```
            uint256 toFree = _profit.add(_debtPayment);

            //freed is math.min(wantBalance, toFree)
            (uint256 freed, ) = liquidatePosition(toFree);

            if (_profit.add(_debtPayment) > freed) { // ***
                if (_debtPayment > freed) {
                    _debtPayment = freed;
                    _profit = 0;
                } else {
                    _profit = freed - _debtPayment; // *** remove
                }
            }
```

Recommendation

use `toFree` in place of `_profit.add(_debtPayment)` and remove `else { ... }`

8. remove `tradesEnabled` (datapunk)

Proof of concept

`tradesEnabled` appeared twice in StrategyConvexFactoryClonable.sol#L153, StrategyConvexFactoryClonable.sol#L301 It does not seem to have any practical usage. In case there is a usage, `_removeTradeFactoryPermissions()` should correspondingly mark

`tradesEnabled = false;`.

Recommendation

Remove `tradesEnabled`

Developer Response

It has been removed, thanks.

9. Iterating through `rewardsTokens` in `_setUpTradeFactory` and `_removeTradeFactoryPermissions` (verypoor)

Proof of concept

If `rewardsTokens` is expected to be more than 1 on average (StrategyConvexFactoryClonable.sol#L291 and StrategyConvexFactoryClonable.sol#L662), and the function always iterate through all reward tokens, it would save gas to copy the `rewardsTokens` to a memory variable. Consequent access of the `rewardsTokens` and its length would only need to load from memory. Since the rewardTokens is unbounded in RewardContract, this can save more gas when there are a lot of reward tokens.

Recommendation

Copy `rewardsTokens` to memory: `address[] memory _rewardsTokens = rewardsTokens;`

10. Redundant external call when using `staticcall` (verypoor)

Proof of concept

BalancerGlobal.sol#L370 did not make use of the returned data from low level `staticcall`, instead, the function makes one extra call to `registry.latestVault(lptoken);` in the success case.

Recommendation

It's both a better practice as well as a gas saving to use `try/catch` syntax introduced since Solidity 6.0. It removes the need to encode calldata and decode returned data, it also gets rid of the external call in case of success. Alternatively, the function can make use of the returned data from `staticcall` (`staticcall` returns data after Solidity 5.0).

Informational Findings

1. Upgrade Pragma (devtooligan)

The pragma used in these contracts is 0.6.12. There have been significant changes made to Solidity since which include new safety features, bug fixes, and optimizations.

Impact

By virtue of using an older pragma, there is an added level of complexity and risk. For purposes of this review, we have been using the security assumptions that go along with this older pragma. But having to abide by these older standards in and of itself is an added step of complexity. For more casual observers, reviewer, integrators, and other stakeholders in the code base, it is challenging to remember all of the security assumption and changes made since then. Additionally, changes have been made to the compiler which would likely result in runtime gas savings.

Recommendation

Upgrade to pragma 0.8.x. Carefully review the features available in 0.8.0+ to determine which specific pragma to use. To optimize for gas savings, consider using the most recent 0.8.15 in conjunction with –via-ir pipeline.

Developer Response

2. Informational - Missing events in BalancerGlobal & StrategyConvexFactoryClonable (Benjamin Samuels, pashov, datapunk)

The BalancerGlobal & StrategyConvexFactoryClonable contracts does not emit events when several important functions are called. These function should emit events to help index the contract's behavior off-chain from Graph Protocol or Yearn Exporter.

If there is no need to index this activity off-chain, then this finding can be ignored.

- setOwner
- acceptOwner
- setAuraPoolManager
- setRegistry
- setBooster
- setGovernance
- setManagement
- setGuardian

- setTreasury
- setKeeper
- setHealthcheck
- setTradeFactory
- setDepositLimit
- setAuraStratImplementation
- setKeepCRV
- setKeepCVX
- setHarvestProfitMaxInUsdt
- setPerformanceFee
- setManagementFee
- setHarvestProfitNeeded
- updateLocalKeepCrvs

Developer Response

3. ERC20.safeApprove is deprecated (pashov)

Proof of concept

StrategyConvexFactoryClonable has several occurrences of the `safeApprove` method calls. This method has been deprecated by OpenZeppelin and its usage is discouraged.

Impact

If OpenZeppelin decide to remove the method since is deprecated, this can result in the code not compiling. Also it wastes gas and gives you a bit of a false sense of security.

Recommendation

In this particular use case it is perfectly fine to use the normal `approve` method from ERC20.

Developer Response

Fixed

4. Informational- incorrect comment (Jib)

Proof of concept

L505 comment on turnOffRewards() says it will set the allowance on the router to 0. This doesnt happen in turnOffRewards

Impact

Missed logic or incorrect comment

Recommendation

Fix the comment to remove the zero allowance set comment or add in the allowance reset to the function.

Developer Response

Comment removed.

5. Informational- hardcoded addresses (verypoor)

Proof of concept

Hard coded addresses in mulitple places: StrategyConvexFactoryClonable.sol and BalancerGlobal.sol.

Impact

When deploying on different chains, and the addresses are not changed accordingly, it can lead to unexpected contract behavior, and in some cases, misconfigured ownership.

Recommendation

Refactor hardcoded addresses to constructors and setters. Extract the addresses to configuration files, which can be used to deploy the contracts.

Developer Response

Hardcoded addresses were a design decision to increase immutability where reasonable and possible.

Out of Scope findings

1. Medium - `VaultRegistry.latestVault()` functions can consume all gas (blockdev)

Gas consumption of copying a storage array is proportional to the size of the array. Once its size becomes big enough, it can result in unwanted reverts due to out of gas issues.

Proof of concept

Vault registry is a proxy and 0xc3efbfdb50cf06e8e5bb623af28678d72caeafea is its current implementation.

It has 2 `latestVault()` functions:

```
function latestVault(address _token) external view returns (address) {
    address[] memory tokenVaults = vaults[_token][VaultType.DEFAULT]; // dev: no
vault for token
    return tokenVaults[tokenVaults.length - 1]; // dev: no vault for token
}

function latestVault(address _token, VaultType _type)
    external
    view
    returns (address)
{
    address[] memory tokenVaults = vaults[_token][_type];
    return tokenVaults[tokenVaults.length - 1]; // dev: no vault for token
}
```

Both the functions copy the storage array in memory.

Impact

Medium. Due to the gas intensive array copying operation, the transactions dependent on these functions can revert. For the audit scope, this means the user will be not be able to create a vault.

However, since Vault registry is a proxy, the implementation can be upgraded, so the risk is not high.

Recommendation

Both of these functions return the last element of an array and revert if the array is empty. To do that, we can skip copying the storage array in memory and directly return the last element by fetching the length first. Update these functions as follows:

```
function latestVault(address _token) external view returns (address) {
    uint256 length = vaults[_token][VaultType.DEFAULT].length; // dev: no vault for
token
    return vaults[_token][VaultType.DEFAULT][length - 1]; // dev: no vault for token
}

function latestVault(address _token, VaultType _type)
    external
    view
    returns (address)
{
    uint256 length = vaults[_token][_type].length; // dev: no vault for token
    return vaults[_token][_type][length - 1]; // dev: no vault for token
}
```

Now these functions consume same amount of gas irrespective of the array length.

Final remarks

Peep

A well written contract with little critical issues. Good news being, some of the more concerning parts of code stability actually happens to be with regards to governance params on CRV / CVX, which is fine. On review, I'm happy that wavey (contract dev) was able to patch many things mentioned in the report. No code review was done on the patches, however.

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, a residents program, and a guest auditor 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). Guest auditors are experts with a track record in the security space who temporarily assist with the review efforts.