yAcademy Joint Strategy review

Review Resources:
None provided beyond the code repository

Residents:

- NibblerExpress

Table of Contents

tags: `Review` `yAcademy`

Review Summary

Joint Strategy

The purpose of the Joint Strategy is to create a set of smart contracts which take equal capital from two strategies, create an LP, and farm. Other than NoHedgeJoint.sol and SolidexJoint.sol, the contracts create a hedge position to offset impermanent loss. The ProviderStrategy contract interfaces with the vaults, and each instance receives funds from a corresponding vault and deposits the funds into the joint contract that manages the LP and hedge. There are four types of hedge contracts: HegicJoint.sol, HedgilJoint.sol, HedgilV2Joint.sol, and NoHedgeJoint.sol, and four corresponding DEX contracts SushiJoint.sol, SpiritJoint.sol, SpookyJoint.sol, and SolidexJoint.sol. The Yearn multisig will manage the contracts. The LP and hedge positions will be manually opened during suitable market conditions.

The main branch of the Joint Strategy Repo was reviewed over 34 days, 3 of which were used to create an initial overview of the contract. The code review was performed between April 19 and May 23, 2022. The code was reviewed by 2 residents for a total of 38 man hours (NibblerExpress: 38 hours). Review of the repository was limited to one specific commit.

Scope

Code Repo
Commit

The commit reviewed was f88f53ec676cedf46ef8fc3e5511872561586d51. The review covered the entire repository at this specific commit but focused on the contracts directory.

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 contract agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Access was limited to governance, vault managers, and keepers (other than view functions and cloning). Other users interact with the contracts through the Yearn vaults rather than accessing directly.
Mathematics	Average	Open Zeppelin math library is used to perform checked math. There is no unchecked math, and no low-level bitwise operations are performed. There is some complex math to balance the profits of both vaults and to compute how many options to purchase.
Complexity	Low	The contracts and functions were well structured, but the functionality is complex and comments and documentation are limited. There is complex math with limited documentation.
Libraries	Average	External libraries are limited to basic Open Zeppelin and well known libraries and interfaces, such as Uniswap and MasterChef.
Decentralization	Low	Access controls provide significant power to governance and vault managers to change the state of the contracts. The risk will be reduced if a multisig is used for governance. Protections are included to prevent removal of vault tokens.
Code stability	Good	Changes were reviewed at a specific commit and the scope was not expanded after the review was started. It did not appear that changes were made to the repository while review was occurring.
Documentation	Low	Comments were included at some important locations but were lacking in many other locations. There appeared to be undocumented assumptions about which tokens would be used. There was some basic documentation outside the code of how the various contracts would function but with limited detail. Documentation did not include derivations for complex math solutions.
Monitoring	Low	No events were emitted other than for cloning in the main contracts for investing and hedging. Some require statements did not include explanations for why they reverted.
Testing and verification	Average	Most contract functionality was covered with basic tests. There was little testing of edge or corner cases and no testing of attack vectors (e.g., sandwich attacks).

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 - No minimum out on sandwichable calls (NibblerExpress)

There are calls to swap tokens using Uniswap or to add tokens to a liquidity position. The comments acknowledge that the calls are sandwichable but does not set a minimum amount out.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L535
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L622
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L645
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L730
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L182
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L212
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L231
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L251

Impact

High. A sandwich attack may result in loss of funds.

Recommendation

Create a settable variable that specifies maximum slippage relative to an oracle price. The variable can always be adjusted when a large amount of slippage is tolerable. `_isWithinRange()` offers some protection to `createLP()` and `_closePosition()` in Joint.sol but not to `sellCapital()` nor to any functions in SolidexJoint.sol.

Developer Response

Our concern with setting a `minOut` from an oracle is that it may prevent closing the position in certain cicrumstances. We will reevaluate the risk/reward of doing so in the next iteration. In the interium, we always use private relays when doing sandwichable actions. The remaining surface for sandwich attachs is mainly uncle bandit attacks.

Medium Findings

1. Medium - Lack of stale data check for oracle (NibblerExpress)

There is no check of round or timestamp in `getCurrentPrice()`. The contract will not know if it is getting stale price data.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/libraries/LPHedgingLib.sol#L68

Impact

Medium. The contract may buy the options at a bad price and be more susceptible to front running.

Recommendation

Revert if the round or timestamp indicates stale price data:

```
(uint80 round, int256 answer, , uint256 time, uint80 answeredRound) = pp.latestRoundData();
require(answeredRound >= round, "Stale price: round");
require(time != 0, "Stale price: time");
return uint256(answer);
```

Developers should also consider L5.

Developer Response

We will include staleness checks in future iterations.

Low Findings

1. Low - Sweep function does not protect reward token (NibblerExpress)

The sweep function in Joint.sol includes require statements on lines 762 and 763 that prevent the sweeping of `tokenA` and `tokenB`. There is no such protection for the reward token, but the function does include an `onlyGovernance` modifier. There is a trade off here between the danger of governance sweeping the reward token versus a router or other failure preventing conversion of the reward token to `tokenA` or `tokenB`. (This is also true of the pair token, but the pair token should always be staked. Overridden functions that fail to stake could introduce a low impact vulnerability.)

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L761

Impact

Low. It is unlikely that governance will sweep the reward token, but it seems inconsistent to protect `tokenA` and `tokenB` not the reward token.

Recommendation

Developers should evaluate the trade off mentioned above and decide which they prefer. Developers should also consider I1.

Developer Response

Acknowledged

2. Low - DoS of `openPosition` by front running (NibblerExpress)

The `openPosition()` function checks that pair is zero, stake is zero, `investedA` is zero, and `investedB` is zero. Pair or stake can easily be made non-zero by sending of the pair token to the contract (in the former case) or using the staking contract to deposit stake token for the user. If front running is possible, DoS is possible by depositing tokens before every attempt to open a position. Even without front running, an attacker can force usage of the manual function to remove the tokens.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L293

Impact

Low. The DoS will prevent opening of positions but will not prevent retrieval of funds.

Recommendation

It is not clear that all the listed checks need to be made. It may be sufficient to check `investedA` and `investedB` are zero.

Developer Response

This attack vector's net result would be giving the strategy a dust sized profit. It's unclear if any economic damage could be enacted with this type of attack.

3. Low - Contract lock-up after manual position close (NibblerExpress)

Much of the functionality of the `closePositionReturnFunds()` function can be executed using the manual functions (e.g., manual calls to `_closePosition()`, any needed/desired swaps, and `_returnLooseToProviders()`). None of these functions change the values of `investedA` nor `investedB`. Because `investedA` and `investedB` need to be zero to open a new position, new positions cannot be opened. `closePositionReturnFunds()` will need to be called to clear the values, which will cost extra gas.

Proof of concept

`investedA` and `investedB` are only set here:
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L264

Impact

Low. The problem can be remedied by calling `closePositionReturnFunds()`, but it will cost extra gas.

Recommendation

Set `investedA` and `investedB` to zero in `_closePosition()`.

Developer Response

Acknowledged

4. Low - Replace transfer with safeTransfer (NibblerExpress)

Although Joint.sol uses `safeTransfer()`, ProviderStrategy.sol is still using `transfer()` with the return value unchecked.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/ProviderStrategy.sol#L152

Impact

Low. Transfer can fail silently.

Recommendation

Replace `transfer()` with `safeTransfer()` or check the return value.

Developer Response

Acknowledged

5. Low - `_isWithinRange()` offers limited protection when oracle price is bad (NibblerExpress)

`_isWithinRange()` compares the spot price to the orace price to determine whether the price is manipulated. As discussed in M1, there are no checks as to whether the oracle price is current. The spot price can be manipulated when the oracle price is bad and vice versa.

Proof of concept

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HegicJoint.sol#L239
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilJoint.sol#L248
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilV2Joint.sol#L248

Impact

Low. Attacks will need to manipulate two prices or rely on an oracle failure.

Recommendation

Confirm the oracle price is current as discussed in M1. Manually specify a target strike price to detect price manipulation.

Developer Response

Acknowledged

Gas Savings Findings

1. Gas - Remove unused function calls (NibblerExpress)

There are cases where a function is called unnecessarily. The function call could be omitted to save gas.

Proof of concept

The following examples were found:

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L355
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L383
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilJoint.sol#L187
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilV2Joint.sol#L187

Impact

Gas savings

Recommendation

Remove the unnecessary function calls.

Developer Response

Acknowledged

2. Gas - Remove unused return variables (NibblerExpress)

There are cases where a variable is created to receive a variable returned by a function and that variable is unused. The return value could be ignored to save gas.

Proof of concept

The following examples were found:

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L255

Impact

Gas savings

Recommendation

Remove the variable and do not receive the return value.

Developer Response

Acknowledged

3. Gas - Change function mutability (NibblerExpress)

The functions in Joint.sol for checking authorized roles can be set to view. In LPHedgingLib.sol, `getCurrentPrice()` can be set to view, and `getCallAmount()` and `getPutAmount()` can be set to pure.

Proof of concept

The following functions could be changed:

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L101
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L107
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L113

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L119
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/libraries/LPHedgingLib.sol#L68
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/libraries/LPHedgingLib.sol#L185
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/libraries/LPHedgingLib.sol#L194

Impact

Gas savings

Recommendation

Add view to the function header for the Joint.sol functions checking authorized roles and to `getCurrentPrice()` and change view to pure for `getCallAmount()` and `getPutAmount()`.

Developer Response

Acknowledged

4. Gas - Remove never true if statement (NibblerExpress)

There is an if statement at line 123 in ProviderStrategy.sol that definitionally will always be false.

Proof of concept

`uint256 amountRequired = _debtOutstanding.add(_profit);`
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/ProviderStrategy.sol#L105

`_debtPayment = _debtOutstanding;` (Assuming else branch is executed.)
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/ProviderStrategy.sol#L121

`if (amountRequired.sub(_debtPayment) < _profit)`
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/ProviderStrategy.sol#L123

By definition `amountRequired` minus `_debtPayment` is equal to `_profit`.

Impact

Gas savings

Recommendation

The if statement at line 123 and the operations inside the if statement can be removed.

Developer Response

Acknowledged

5. Gas - Rearrange if conditions (NibblerExpress)

The `_autoProtect()` function is checked before the `autoProtectionDisabled` variable at Line 217. Checking the `_autoProtect()` function is more gas intensive than checking the `autoProtectionDisabled` variable, so the developers should consider checking the latter first. If the `autoProtectionDisabled` variable will be false the vast majority of the time, the current ordering would make sense.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L217

Impact

Gas savings

Recommendation

Reverse the conditions in the if statement.

Developer Response

Acknowledged

6. Gas - Remove extraneous calculations in `balanceOfTokensInLP()` (NibblerExpress)

The function `balanceOfTokensInLP()` does a multiplication by `pairPrecision` followed by a division by `pairPrecision`.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L704

Impact

Gas savings

Recommendation

Remove the multiplication and division by `pairPrecision` as well as the calculation of `pairPrecision`.

```
        (uint256 reserveA, uint256 reserveB) = getReserves();
        uint256 lpBal = balanceOfStake().add(balanceOfPair());
        _balanceA = reserveA.mul(lpBal).div(pair.totalSupply());
        _balanceB = reserveB.mul(lpBal).div(pair.totalSupply());
```

Developer Response

Acknowledged

7. Gas - Use != 0 for gas savings (NibblerExpress)

Using `> 0` is less 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

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/ProviderStrategy.sol#L136
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/ProviderStrategy.sol#L151
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L174
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L663
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L668
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HegicJoint.sol#L180
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SushiJoint.sol#L150
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SushiJoint.sol#L157
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L148
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L155
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SpiritJoint.sol#L145
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SpiritJoint.sol#L152
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SpookyJoint.sol#L145
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SpookyJoint.sol#L152

Impact

Gas savings

Recommendation

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

Developer Response

8. Gas - Using simple comparison (NibblerExpress)

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

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L608

Impact

Gas savings

Recommendation

Replace compound comparison operators with simple ones for gas savings

Developer Response

9. Gas - Remove duplicate comparison (NibblerExpress)

The code checks the conditions `is_weth || is_internal` twice. Gas could be saved by just checking the condition once.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L576

Impact

Gas savings

Recommendation

Developer Response

10. Gas - Remove never true if statement (NibblerExpress)

There is an if statement in `calculateSellToBalance()` at line 431 in Joint.sol that will always be false or could be implemented more efficiently elsewhere.

Proof of concept

When `calculateSellToBalance()` is called from `closePositionReturnFunds()`, `closePositionReturnFunds()` has already checked whether `investedA` or `investedB` is zero. The if statement will always be false.
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L431
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L376

`calculateSellToBalance()` is also called from `estimatedTotalAssetsAfterBalance()`. The if statement could be moved into `estimatedTotalAssetsAfterBalance()` to save gas. If `investedA` and `investedB` are always both zero or both nonzero (see I2 and L3), much of the logic in `estimatedTotalAssetsAfterBalance()` could be skipped when `investedA` and/or `investedB` is zero.

Impact

Gas savings

Recommendation

Remove the if statement from line 431. Consider whether to add an if statement to `estimatedTotalAssetsAfterBalance()`.

Developer Response

11. Gas - Simpler first estimate of sell amount (NibblerExpress)

There could be a simpler first estimate of the sell amount in `_calculateSellToBalance()` to save calculations and avoid having to retrieve and pass the precision variable.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L474
The `_calculateSellToBalance()` code could be modified to resemble the below untested code:

```
        //First time to approximate
        uint256 initialAmountIn =
            current0.mul(starting1).sub(starting0.mul(current1))
            .div(starting1 + starting0.mul(reserve1).div(reserve0));
        uint256 initialAmountOut;

        // Second time to account for price impact and fees
        initialAmountOut = UniswapV2Library
            .getAmountOut(initialAmountIn, reserve0, reserve1);
        _sellAmount =
            current0.mul(starting1).sub(starting0.mul(current1))
            .div(starting1 + starting0.mul(initialAmountOut).div(initialAmountIn));
```

A similar change could be made to SolidexJoint.sol:
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L416

Impact

Gas savings

Recommendation

Revise `_calculateSellToBalance()` to have code similar to what is listed above. (Note I also removed the `if (_sellAmount == 0)` because this situation should have been caught by earlier checks in `calculateSellToBalance()`.)

Developer Response

12. Gas - Declare functions external for gas savings (NibblerExpress)

Several functions can be declared as external instead of public to save gas.

Proof of concept

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/ProviderStrategy.sol#L220
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L167
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L171
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L403
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/NoHedgeJoint.sol#L36

Impact

Gas savings

Recommendation

Declare functions as external functions for gas savings (including overriding functions).

Developer Response

13. Gas - Simplify `getTokenOutPath` (NibblerExpress)

In ProviderStrategy.sol, `tokenToWant()` is only called by `ethToWant()`, so much of the code in `getTokenOutPath()` can be simplified. The path is always directly from WETH to want, and `getTokenOutPath()` can simply return this path.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/ProviderStrategy.sol#L234

Impact

Gas savings

Recommendation

Set path to be from WETH to want in `getTokenOutPath()` without the additional logic.

Developer Response

14. Gas - Remove unnecessary calculation (NibblerExpress)

In `prepareReturn()`, the `_debtOutstanding` should never exceed the `_totalDebt`. Because `amountAvailable` is `totalAssets`, `_profit = totalAssets.sub(_totalDebt);`, and `amountRequired = _debtOutstanding.add(_profit);`, `amountRequired` should never exceed `amountAvailable` unless there is a loss. This means that `_profit` does not need to be recalculated when `amountRequired > amountAvailable`.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/ProviderStrategy.sol#L110
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/ProviderStrategy.sol#L118

Impact

Gas savings

Recommendation

Remove updates to `_profit` when `amountRequired > amountAvailable`.

Developer Response

15. Gas - Move up require statement to revert sooner (NibblerExpress)

`hedgeLP()` includes a require statement that could be moved earlier in the code to use less gas when a transaction reverts.

Proof of concept

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilJoint.sol#L190
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilV2Joint.sol#L190
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HegicJoint.sol#L186

Impact

Gas savings

Recommendation

Move require statement to earlier in the code.

Developer Response

Acknowledged

16. Gas - Unnecessary hedging code in SolidexJoint (NibblerExpress)

SolidexJoint.sol extends NoHedgeJoint.sol, so it includes unnecessary calculations of hedging amounts.

Proof of concept

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L182
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L204
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L330

Impact

Gas savings

Recommendation

Remove unnecessary function calls involving hedging.

Developer Response

17. Gas - Unnecessary check of path length (NibblerExpress)

In SolidexJoint.sol, `getTokenOutPathSolid()` checks whether `_path.length > 1`, but `_path` is sized to 2 or 3 and should never be 1.

Proof of concept

Path length set here:
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L272

Check of path length is here:
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L281

Impact

Gas savings

Recommendation

Remove unnecessary check of path length.

Developer Response

18. Gas - Use pre-increment to save gas (NibblerExpress)

Using a pre-increment (`++i`) is more efficient than using a post-increment (`i++`).

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L283

Impact

Gas savings

Recommendation

Change post-increment to pre-increment.

Developer Response

19. Gas - Remove logic from `swapReward()` in SolidexJoint.sol (NibblerExpress)

Rewards are not swapped in SolidexJoint.sol, so token and ratio comparison logic is unnecessary.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L289

Impact

Gas savings

Recommendation

Remove all the logic and `return (0, 0)`.

Developer Response

20. Gas - Use of boolean equality (NibblerExpress)

Boolean values do not need to be compared with true or false.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L308

Impact

Gas savings

Recommendation

Change `tradesEnabled==false` to `!tradesEnabled`.

Developer Response

21. Gas - Unused functions (NibblerExpress)

The internal function `getHedgeStrike()` is not used and can be removed.

Proof of concept

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilJoint.sol#L172
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HegicJoint.sol#L158

Impact

Gas savings

Recommendation

Delete the unused internal function.

Developer Response

22. Gas - Tautology (NibblerExpress)

`_autoProtect()` contains a tautology that can be returned directly.

Proof of concept

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HegicJoint.sol#L280
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilJoint.sol#L294
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilV2Joint.sol#L294

Impact

Gas savings

Recommendation

Use `return timeToMaturity == 0 || timeToMaturity > period.mul(50).div(100)`.

Developer Response

23. Gas - Unnecessary data retrieval and calculations (NibblerExpress)

`getLPInfo()` includes unnecessary data retrievals and calculations for both sides of the pool.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/libraries/LPHedgingLib.sol#L214

Impact

Gas savings

Recommendation

Use code similar to this:

```
        address mainAsset = address(hegicCallOptionsPool.token());
        require(mainAsset == IUniswapV2Pair(lpToken).token0() || mainAsset == IUniswapV2Pair(lpToken).token1(),
"LPtoken not supported");
        uint256 balance = IERC20(mainAsset).balanceOf(address(lpToken));
        uint256 amount = IUniswapV2Pair(lpToken).balanceOf(address(this));
        uint256 totalSupply = IUniswapV2Pair(lpToken).totalSupply();
        q = amount.mul(balance) / totalSupply;
```

Developer Response

24. Gas - Unnecessary constant (NibblerExpress)

The constant `hegicOptionsManager` is unused and can be removed.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/libraries/LPHedgingLib.sol#L38

Impact

Gas savings

Recommendation

Remove the unused constant.

Developer Response

25. Gas - Use Uniswap slippage checks (NibblerExpress)

`removeLiquidityManually()` uses 0 as inputs to the `minAmountOut` for Uniswap's `removeLiquidity()` and then includes additional checks of whether the balance out exceeded what was expected.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L725

Impact

Gas savings

Recommendation

Use the `minAmountOut` rather than additional checks.

Developer Response

26. Gas - Remove multiple SLOADs (NibblerExpress)

Some functions include multiple SLOADs rather than saving gas by using an SLOAD and MSTORE followed by multiple MLOADs. Specifically, `balanceOfPair` is retrieved multiple times in `openPosition()` and `_closePosition()` and `balanceOfStake` is retrieved multiple times in `openPosition()`.

Proof of concept

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L294
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L295
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L312
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L639
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L648

Impact

Gas savings

Recommendation

Store values (e.g., `balanceOfPair` and `balanceOfStake`) to local variables rather than loading repeatedly, or remove repeated loads (e.g., by using recommendation in L2).

Developer Response

Informational Findings

1. Informational - There is no function to update router (NibblerExpress)

The router is set in the constructor, and there is no way to update it.

Proof of concept

Router is set here:
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L145

Impact

Informational. If there is a problem with the router, there is no way to update it in the contract. A new instance of the contract will need to be deployed with the new router. It will likely be possible to recover all tokens from the contract if there is a problem with the router, but they may not be allocated evenly among the providers.

Recommendation

Include a function to update the router.

Developer Response

2. Informational - Potential lock-up if `investedA` or `investedB` is zero and the other is nonzero (NibblerExpress)

Before the contract sets `investedA = investedB = 0`, it checks whether `investedA == 0 || investedB == 0`. If one variable is zero and the other is nonzero, the function will return before setting both values to zero. The contract will lock-up and positions cannot be opened or closed. It should never be true that one variable is zero and the other is not, but this is enforced by the DEX contract. (Uniswap appears to enforce this invariant. Solidly and other DEXs have not been confirmed.) This issue may arise if the contract is used in the future with a DEX not enforcing this invariant.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L222

Impact

Informational. This may be an issue in the future but does not appear to be one at present.

Recommendation

Change the OR (`||`) to AND (`&&`) or change `investedA` and `investedB` to zero in `_closePosition` as suggested in L3.

Developer Response

3. Informational - Trade to WETH may not always be ideal (NibblerExpress)

`swapReward()` always swaps to WETH when one of the tokens is WETH. If the other token has appreciated against WETH, swapping to WETH when closing the position will result in a larger swap back the other way to balance the tokens.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L601

Impact

Informational. There may be a slightly higher gas cost from swapping to WETH and due to the additional code.

Recommendation

Remove the if block that checks whether one of the tokens is WETH and just use the default determination of which way to swap the reward.

Developer Response

4. Informational - `minAmountToSell` not initialized (NibblerExpress)

`minAmountToSell` is not initialized, so the protocol may swap token amounts that are too small to justify the gas.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L254

Impact

Informational. There may be a slightly higher gas costs due to swapping small token amounts.

Recommendation

Set `minAmountToSell` in the constructor/initialization function.

Developer Response

5. Informational - Manual close required when losses are incurred (NibblerExpress)

`closePositionReturnFunds()` reverts if either token experiences a loss beyond the max percentage loss (initialized at 0.1%). If a loss is suffered due to a large and rapid swing in prices, the positions will have to be closed manually.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L269

Impact

Informational. Managers will need to react appropriately when losses are incurred.

Recommendation

Include a boolean input to the function that allows you to skip the check (e.g., skip the check when `liquidateAllPositions()` calls `closePositionReturnFunds()` but not when `prepareReturn()` calls `closePositionReturnFunds()`).

Developer Response

6. Informational - Typos (NibblerExpress)

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

Proof of concept

Sandwidched:

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L717
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L718
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L739
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L740
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L241
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L242

Impact

Informational

Recommendation

Fix typos.

Developer Response

7. Informational - Hardcoding of addresses (NibblerExpress)

The address of the `tradeFactory` is hardcoded in ySwapper.sol and Joint.sol, as are the addresses of `SEX` and `SOLID_SEX`.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/ySwapper.sol#L17
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L149
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L24
https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/DEXes/SolidexJoint.sol#L25

Impact

Informational

Recommendation

Pass the address as an input to the constructor.

Developer Response

8. Informational - Liquidate position creates illusory losses (NibblerExpress)

`liquidatePosition()` returns a loss if there is not enough balance in ProviderStrategy.sol to cover the `amountNeeded`.

Proof of concept

https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/ProviderStrategy.sol#L158

Impact

Informational

Recommendation

`liquidatePosition()` could check whether the Joint contract has loose want. The Yearn vault could be modified to account for illusory losses.

Developer Response

9. Informational - NoHedgeJoint.sol does not detect high impermanent loss (NibblerExpress)

In NoHedgeJoint.sol, `shouldEndEpoch()` and `_autoProtect()` always return false.

Proof of concept

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/NoHedgeJoint.sol#L59
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/NoHedgeJoint.sol#L64

Impact

Informational

Recommendation

`shouldEndEpoch()` and `_autoProtect()` should indicate when there has been a high level of impermanent loss.

Developer Response

10. Informational - Different treatment of time to maturity (NibblerExpress)

In LPHedgingLib.sol, `closeHedge()` assumes that the expiration is the same for call and put options whereas `getTimeToMaturity()` compares expiration times and chooses the earlier.

Proof of concept

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/libraries/LPHedgingLib.sol#L156
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/libraries/LPHedgingLib.sol#L254

Impact

Informational

Recommendation

Consistent assumptions should be used to ensure that contract does not create unexpected results (e.g., if forked).

Developer Response

11. Informational - Require statements do not report reason for revert (NibblerExpress)

Several require statements do not state the reason for revert.

Proof of concept

Examples are too numerous to list them all. A few examples:

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L85
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L293
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilJoint.sol#L190
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilJoint.sol#L198

Impact

Informational

Recommendation

State the reason for revert to make it easier to understand why a transaction failed.

Developer Response

12. Informational - `closeHedgeManually()` does not take hedge ID as input (NibblerExpress)

In HedgilJoint.sol and Hedgilv2Joint.sol, `closeHedgeManually()` does not accept the hedge ID as an input, so the hedge cannot be closed after `resetHedge()` is used. In contrast, HegicJoint.sol allows the `callID` and `putID` to be specified.

Proof of concept

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilJoint.sol#L164
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilV2Joint.sol#L168

Impact

Informational

Recommendation

Take the hedge ID as an input in the manner done in HegicJoint.sol.

Developer Response

13. Informational - Assumption hedge is always paid in `tokenB` (NibblerExpress)

In HedgilJoint.sol and Hedgilv2Joint.sol, it is assumed that `tokenB` is always used to pay for the hedge. This assumption relies on the deployment to use a particular token for `tokenB`.

Proof of concept

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilJoint.sol#L80
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilV2Joint.sol#L79

Impact

Informational

Recommendation

Ensure deployment scripts use appropriate token.

Developer Response

14. Informational - Hardcoding of constant (NibblerExpress)

The limit for hedging period is hard coded in the hedging contracts.

Proof of concept

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HegicJoint.sol#L141
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilJoint.sol#L152
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Hedges/HedgilV2Joint.sol#L156

Impact

Informational

Recommendation

Create a constant or settable variable that specifies the upper limit for the hedging period.

Developer Response

15. Informational - Now is deprecated (NibblerExpress)

The code uses `now` rather than `block.timestamp`. This should be updated if newer versions of solidity are used.

Proof of concept

- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L547
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L627
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L652
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L737
- https://github.com/fp-crypto/joint-strategy/blob/f88f53ec676cedf46ef8fc3e5511872561586d51/contracts/Joint.sol#L756

Impact

Informational

Recommendation

Replace `now` with `block.timestamp`.

Developer Response

Final remarks

NibblerExpress

The code was well organized, and the basic structure of Yearn vaults and strategies leaves a limited attack surface. The main vulnerabilities result from interactions with exchanges and oracles. The code does mitigate against some manipulation of exchanges or oracles, but there could be additional protections. The comments suggest that private relays will also be used to mitigate front running or sandwich attacks. For purposes of this review, I assumed that a private relay would not be used because the code cannot guarantee this will be true. There are some situations that could result in some functions being unavailable due to variables being in an unexpected state or due to denial of service attacks. The code includes work arounds for all such situations, so findings were generally low or informational. Governance/vault managers/keepers will need to understand how to use the work arounds. In a stressful situation due to high volatility, additional losses may be incurred while governance/vault managers/keepers determine the work around. There appear to be many assumptions about how the code will be used. Any forks or updated versions used with different tokens or exchanges should be carefully reviewed to ensure the assumptions remain true.

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`