

# yAcademy Ohm review

## Review Resources:

[Requirements Doc](#)

[Support Doc](#)

## Residents:

- NibblerExpress
- Sjkellyjr
- Engin33r

## Table of Contents

### yAcademy Ohm review

Table of Contents

Review Summary

Scope

Code Evaluation Matrix

Findings Explanation

High Findings

1. staking.unstake() should set rebasing bool to true (engin33r)

Proof of concept

Impact

Recommendation

Remediation Status

2. Sandwich attack risk in YieldStreamer.sol (engin33r)

Proof of concept

Impact

Recommendation

Remediation Status

Medium Findings

1. Denial of service in upkeep "for" loop in YieldStreamer.sol (sjkelleyjr)

Proof of concept

Impact

Recommendation  
Remediation Status

#### Low Findings

1. `donatedTo` and `depositsTo` only return the first of N possible donation or deposit mappings (sjkelleyjr)

Proof of concept  
Impact  
Recommendation  
Remediation Status

2. `_redeemAll()` does not delete `recipientLookup` (sjkelleyjr)

Proof of concept  
Impact  
Recommendation  
Remediation Status

#### Gas Savings Findings

1. Declare function external for gas savings (sjkelleyjr)

Proof of concept  
Impact  
Proof of concept  
Remediation Status

2. Use `==` for gas savings (sjkelleyjr)

Proof of concept  
Impact  
Recommendation  
Remediation Status

3. Simplify math for gas savings (engn33r)

Proof of concept  
Impact  
Recommendation  
Remediation Status

4. Use `msg.sender` for gas savings (engn33r)

Proof of concept  
Impact  
Recommendation  
Remediation Status

5. Remove functions to save gas on deployment (engn33r)

Proof of concept

Impact

Recommendation

Remediation Status

6. Gas savings using unchecked (engn33r)

Proof of concept

Impact

Recommendation

Remediation Status

7. Gas savings with memory variable (engn33r)

Proof of concept

Impact

Recommendation

Remediation Status

8. redeemAll repeatedly computes index - 1 (NibblerExpress)

Proof of Concept

Impact

Recommendation

Remediation Status

Informational Findings

1. Miscellaneous Improvement Ideas (engn33r)

Proof of concept

Impact

Recommendation

Remediation Status

2. updateUserMinDaiThreshold assumes Dai is used as the streamToken (sjkelleyjr)

Proof of concept

Impact

Recommendation

Remediation Status

3. Missing calls in scripts/deployAll.js (engn33r)

Proof of concept

Impact

Recommendation

Remediation Status

Final remarks

sjkelleyjr

engn33r

NibblerExpress

About yAcademy

Appendix and FAQ

tags: Review yAcademy

## Review Summary

### Tyche/OlympusGive

The purpose of Tyche/OlympusGive is to empower Ohm/sOHM/gOHM holders to use their OHM yield in different ways.

This family of contracts aims to allow users to donate their yield to multiple users. The YieldStreamer contract allows sOHM or gOHM deposits, and the yield can be withdrawn in the form of a stream token (Dai is the default). In contract, the YieldDirector contract permits gOHM deposits and the gOHM rebases can be donated.

The main branch of the OlympusDAO [Repo](#) was reviewed over 9 days, 3 of which were used to create an initial overview of the contract. The code review was performed between March 20 and March 28, 2022. The code was reviewed by 3 residents for a total of 43 hours (Sjkelleyjr: 17 hours, Engn33r: 16 hours, and NibblerExpress: 10 hours) man hours. The repository was under active development during the review, but the review was limited to one specific [commit](#).

## Scope

[Code Repo](#)

[Commit](#)

The review was limited to the three following contracts:

- YieldSplitter.sol
- YieldDirector.sol
- YieldStreamer.sol

The focal point of the review was the following code modifications:

- <https://github.com/OlympusDAO/olympus-contracts/pull/208>
- <https://github.com/OlympusDAO/olympus-contracts/pull/186>
- <https://github.com/OlympusDAO/olympus-contracts/blob/main/contracts/types/YieldSplitter.sol>

The commit reviewed was f92425ff278b93cfc2a580299924417182da432a

After the findings were presented to the Ohm team, a review of mitigations was performed during April 5-9 based on the first three commit of PR 288:

<https://github.com/OlympusDAO/olympus-contracts/pull/288>

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

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

## Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Modifiers such as onlyGovernor and isInvalidDeposit control conditional logic branches, while elsewhere if statements check whether the msg.sender is the depositor or recipient. Access controls are applied where needed.
Mathematics	Good	Solidity 0.8.10 is used, which provides overflow and underflow protect. No unchecked code exists and no low-level bitwise operations are performed.
Complexity	Good	No assembly is used and the code is clearly written and organized.
Libraries	Good	The libraries used are from trustworthy sources. No external libraries from outside the project

repository are imported.

Decentralization	Average	The onlyGovernor modifier implies some level of centralization, although a multisig is expected to mitigate this.
Code stability	Average	Changes were reviewed at a specific commit, additional code development occurred, but the scope was not expanded after the review was started. However, development was ongoing when the review was performed so the code was not frozen, which means deployed code may vary from what was reviewed.
Documentation	Average	Documentation for the project exists and the code had natspec comments for nearly all functions. However, no documentation existed in the official Olympus DAO GitBook documentation and some natspec documentation could be made clearer with standardized terminology across contracts.
Monitoring	Average	Events were added to most important functions that modified state variables, but not the yield withdraw or variable update functions in YieldStreamer.
Testing and verification	Low	The findings indicate that the test coverage could be improved to better handle edge cases. Improved unit tests could enhance consistency and better check the value of all state variables after operations are performed.

## 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 controll/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. `staking.unstake()` should set rebasing bool to true (engn33r)

The YieldStreamer contract passes an incorrect parameter to `staking.unstake()`.

#### Proof of concept

The steps that the YieldStreamer functions take to withdraw yield to stream tokens occur in this order

1. Calculate yield in gOHM
2. Unwrap gOHM to sOHM
3. Unstake sOHM to OHM
4. Convert OHM to stream token

The problem is in step 3. The rebasing bool is set to false, which will cause the staking contract to take gOHM from `msg.sender` instead of taking sOHM as the YieldStreamer contract requires. The logic for the rebasing boolean in the staking contract is

```
if (_rebasings) {
    sOHM.safeTransferFrom(msg.sender, address(this), _amount);
    amount_ = amount_.add(bounty);
} else {
    gOHM.burn(msg.sender, _amount); // amount was given in gOHM terms
    amount_ = gOHM.balanceFrom(amount_).add(bounty); // convert amount to
    OHM terms & add bounty
}
```

In order to convert sOHM to OHM in the `staking.unstake()` function, which is required in step 3 of the steps outlined above, the rebasing parameter must be set to true. Otherwise gOHM will be converted instead.

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/Staking.sol#L181>

Currently the rebasing boolean value is set to false in two locations

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldStreamer.sol#L246>

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldStreamer.sol#L322>

## Impact

High, currently gOHM will be subtracted from the YieldStreamer contract twice while sOHM will sit in YieldStreamer unused.

## Recommendation

Change the rebasing input parameter to true, not false

```
staking.unstake(address(this), totalOhmToSwap, false, true);
```

## Remediation Status

The functions relying on `staking.unstake()` have been changed so that this is no longer an issue. Notably, the complex `withdrawYieldInStreamTokens()` function has been removed. A new observation was that comments still exist in `YieldStreamer.sol` that reference `withdrawYieldInStreamTokens` even though the function has been removed. Only one instance of the `staking.unstake()` function call remains, in the `upkeep()` function, where the token conversion process has been simplified so that the original issue is no longer valid.

## 2. Sandwich attack risk in YieldStreamer.sol (engn33r)

`YieldStreamer.sol` needs to swap Ohm for the streaming token. To do this, the `swapExactTokensForTokens` function from Sushiswap is used. The `swapExactTokensForTokens` requires a `amountOutMin` value be set to reduce slippage and prevent sandwich attacks. In the `YieldStreamer.sol` contract, the `amountOutMin` value that is set is equivalent to not setting a minimum `amountOutMin`. This makes the swap sandwichable. Even if a private relay like flashbots is used to mitigate the sandwich attack risk, [an uncle bandit attack](#) is still a risk to this operation.



## Proof of concept

This code is how the swap is performed in YieldStreamer.sol in two locations:

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldStreamer.sol#L248-L255>

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldStreamer.sol#L324-L331>

```
uint256[] memory calculatedAmounts =
sushiRouter.getAmountsOut(total0hmToSwap, sushiRouterPath);
uint256[] memory amounts = sushiRouter.swapExactTokensForTokens(
    total0hmToSwap,
    (calculatedAmounts[1] * (1000000 - maxSwapSlippagePercent)) / 1000000,
    sushiRouterPath,
    msg.sender,
    block.timestamp
);
```

The amountOutMin value is the output of `getAmountsOut ( total0hmToSwap, sushiRouterPath )` minus some slippage percent `maxSwapSlippagePercent`.

Examining the code of the `swapExactTokensForTokens ( )` function, we can see how `amountOutMin` is used. There is a check to confirm the result of the exchange is greater than the `amountOutMin`

<https://github.com/sushiswap/sushiswap/blob/56cedd0e06a6cf665083b3a662f9f77b80303ebe/contracts/uniswapv2/UniswapV2Router02.sol#L233>

```
amounts = UniswapV2Library.getAmountsOut(factory, amountIn, path);
require(amounts[amounts.length - 1] >= amountOutMin, 'UniswapV2Router:
INSUFFICIENT_OUTPUT_AMOUNT');
```

But because the `swapExactTokensForTokens ( )` function is using the exact same `getAmountsOut ( )` function that was used in `YieldStreamer.sol` to calculate `amountOutMin` and is calculated in the same transaction, the result is that the two `getAmountsOut ( )` output values will always be equal. So the `amountOutMin` set in `YieldStreamer.sol` serves no purpose because it can never yield a result where the `amountOutMin` value reverts the swap, because `amountOutMin` is `getAmountsOut ( )` minus some slippage, which will always be less than or equal to `getAmountsOut ( )`. The current calculation to provide an `amountOutMin` value is a waste of gas.

From the [Uniswap V2 Router](#) source code, we can see that the `amountOutMin` is calculated as follows:

Even in a scenario where a private relay is used, miners cannot guarantee that a block will not become an uncle block. If the block is uncled, the mempool data becomes public and an uncle bandit attack is possible.

## Impact

High, sandwich attack is expected if the liquidity pool has low liquidity relative to a flash loan or whale

## Recommendation

A comment in the PR discussing the sandwich attack risk indicates that a private relay may be necessary

[https://github.com/OlympusDAO/olympus-contracts/pull/186#discussion\\_r7893346](https://github.com/OlympusDAO/olympus-contracts/pull/186#discussion_r7893346)

If no private relay is used, an external oracle should be used instead of relying on the DEX liquidity pool spot price. [Chainlink](#) has several OHM price feeds.

<https://shouldiusespotpriceasmyoracle.com/>

An additional improvement for the `withdrawYieldInStreamTokens()` function is to add a user-specified `minAmountOut` input parameter. Because a check exists in `withdrawYieldInStreamTokens()` requiring `recipientInfo[id_].recipientAddress == msg.sender`, the function caller is the intended recipient and therefore can be trusted to set a slippage value of their choice. To make this even easier for a user, a view function can be created that calculates this input amount to pass as the `minAmountOut` input parameter. This improvement would not work for `upkeep()` because it can be called by anyone and the caller cannot be trusted to set a safe slippage value.

## Remediation Status

This issue has not been remediated at the time of report completion. An additional concern was added about the possibility of swaps that happen earlier in the activeDepositIds array receiving better swap rates than those later in the array.

# Medium Findings

## 1. Denial of service in upkeep "for" loop in YieldStream.sol (sjkelleyjr)

If `upkeep()` is not run frequently enough, or if the upkeep eligibility of many depositors synchronize, the two for loops in the `upkeep()` function may run out of gas, preventing anyone from running this function until (or if) Ethereum mainnet gas limits are increased. There is no backup function in `YieldStream.sol` that allows for upkeep to be run on individual positions, which would provide a solution if it is not possible to upkeep all eligible positions.

This function can be attacked by a user who wants to perform a denial of service. A user can create many deposits with a `paymentInterval` of 1 second. All of these payments will be upkeep eligible, increasing the amount of gas that the upkeep caller must spend. This attack can bring the gas usage of `upkeep()` over the current 30,000,000 mainnet gas limit. The upkeep function is a critical piece of `YieldStream.sol`, and if it is unavailable the contract will cease to be useful, requiring individuals to withdraw their assets and the Ohm team to redeploy an upgraded contract without this weakness.

### Proof of concept

Based on the current gas limit of 30 million, a malicious user Mallory could cause this denial of service with a few thousand deposits

1. Mallory calls `deposit()` in `YieldStream.sol` a few thousand times with small deposits that are equal to or greater than the value of `minimumTokenThreshold`. This causes the length of `activeDepositIds` to increase. When making these deposits, Mallory sets the `paymentInterval` to a very short amount of time and the `userMinimumAmountThreshold` to a very small value.
2. Mallory, or any user, will call `upkeep()` at a later time. Because `_isUpkeepEligible(currentId)` will return true for all of Mallory's deposits, which number in the thousands, the `upkeep()` function will revert when it runs out of gas. The `upkeep()` function will be unusable, meaning users will not receive their stream tokens on schedule. All users will need to withdraw their deposits to receive the stream tokens.

## **Impact**

Medium, contract out of gas causing denial of service

## **Recommendation**

Add a function to upkeep individual deposits instead of only maintaining a single function to upkeep all deposits.

## **Remediation Status**

This issue has not been remediated at the time of report completion. One recommendation for the `upkeep()` denial of service was to add a function to upkeep individual deposits, but if the `redeemYieldOnBehalfOf()` function is intended to serve this purpose, it needs to add the line  
`currentrecipientInfo.lastUpkeepTimestamp = uint128(block.timestamp);`  
so that this deposit is not looped through on the next `upkeep()` call.

# **Low Findings**

## **1. donatedTo and depositsTo only return the first of N possible donation or deposit mappings (sjkelleyjr)**

There can be multiple `depositInfo` mappings for a given donor or multiple `recipientLookup` mappings for a given donor to recipient.

## **Proof of concept**

<https://github.com/OlympusDAO/olympus-contracts/blob/main/contracts/peripheral/YieldDirector.sol#L195>

<https://github.com/OlympusDAO/olympus-contracts/blob/main/contracts/peripheral/YieldDirector.sol#L256>

## **Impact**

Low, the view functions may return incorrect values.

## **Recommendation**

## Recommendation

Return a sum of the total mappings as is done in the other view functions, or ensure the same mapping between donor and recipient can only be created one time when deposit is called.

## Remediation Status

This finding was successfully resolved using the provided recommendation.

## 2. `_redeemAll()` does not delete `recipientLookup` (sjkelleyjr)

In `YieldDirector.sol`, the two mappings `recipientIds` and `recipientLookup` allow for `recipient -> depositId` and `depositId -> recipient` lookups respectively. The two mappings must be modified at the same time and in the same way to maintain relevance. For example, in the `_redeem()` function, when the `principalAmount` of a deposit is zero, the function calls `receiptIds.pop();` and `delete recipientLookup[depositId_]` to remove this `depositId` from the mappings. But in the related `_redeemAll()` function, there is no deletion of the `recipientLookup[depositId_]` value. This can result in the `recipientLookup` mapping storing unnecessary data. This is mostly a gas savings issue, but the extra data in the array may lead to an edge case logic error, though none was identified.

## Proof of concept

In the `_redeem()` function, `receiptIds.pop();` and `delete recipientLookup[depositId_]` are called

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldDirector.sol#L429-L455>

But in the similar `_redeemAll()` function, there is no `delete recipientLookup[depositId_];` call.

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldDirector.sol#L460-L488>

## Impact

## Impact

Low, the recipientLookup mapping will still contain depositId -> recipient mappings even if this is zero principal, but no direct security vulnerability from this data was found.

## Recommendation

Modify the line of code that calls `receiptIds.pop()`; in `_redeemAll()` to the following:

```
delete recipientLookup[receiptIds[index - 1]];
receiptIds.pop();
```

## Remediation Status

This finding was successfully resolved using the provided recommendation.

# Gas Savings Findings

## 1. Declare function external for gas savings (sjkelleyjr)

The `totalRedeemableBalance` function of `YieldDirector.sol` should be declared external, not public, for gas savings.

## Proof of concept

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldDirector.sol#L315>

## Impact

Gas savings

## Proof of concept

Recommendation

Declare `totalRedeemableBalance` as an external function, not a public function

## Remediation Status

## REFERENCES

This finding was successfully resolved using the provided recommendation.

## 2. Use == for gas savings (sjkelleyjr)

The three modifiers of YieldDirector.sol use a check for `amount <= 0`, but the check can be changed to `amount == 0` for gas savings.

There's another related gas optimization. the if/else statement in the `_withdraw()` function of `YieldDirector.sol` can replace `>` with `<` if the if/else clauses are switched around.

The same gas optimization can be used in the if/else statement of the `withdrawPrincipal()` function in `YieldStreamer.sol`.

## Proof of concept

`<=` is used instead of `==` in three modifiers

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldDirector.sol#L71-L81>

$\geq$  can be simplified to  $<$  by flipping the if/else clauses in two locations

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldDirector.sol#L415>

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldStreamer.sol#L163>

An example of flipping the if/else clauses is shown. Here is the original code:

```
if (amount_ >= _toAgnostic(depositInfo[depositId_].principalAmount)) {
    amountWithdrawn = _withdrawAllPrincipal(depositId_, msg.sender);
} else {
    _withdrawPrincipal(depositId_, amount_, msg.sender);
    amountWithdrawn = amount_;
}
```

And here is the modified code with a minor gas savings. This specific gas savings may be negated if the `_withdrawAllPrincipal` clause is used most of the time:

```
if (amount_ < _toAgnostic(depositInto[depositId_].principalAmount)) {
    _withdrawPrincipal(depositId_, amount_, msg.sender);
    amountWithdrawn = amount_;
} else {
    amountWithdrawn = _withdrawAllPrincipal(depositId_, msg.sender);
}
```

## Impact

Gas savings

## Recommendation

Use ==, <, or > instead of ≤ or ≥ whenever possible for gas savings

## Remediation Status

This finding was successfully resolved using the provided recommendation.

## 3. Simplify math for gas savings (engn33r)

Line 128 and 129 of YieldSplitter.sol reads

```
amountRedeemed = _getOutstandingYield(userDeposit.principalAmount,
userDeposit.agnosticAmount);
userDeposit.agnosticAmount = _toAgnostic(userDeposit.principalAmount);
```

Instead of calling `_toAgnostic()`, this can be simplified to save gas with

```
amountRedeemed = _getOutstandingYield(userDeposit.principalAmount,
userDeposit.agnosticAmount);
userDeposit.agnosticAmount = userDeposit.agnosticAmount - amountRedeemed;
```

## Proof of concept

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/types/YieldSplitter.sol#L129>

## Impact



-----  
Gas savings

### **Recommendation**

Simplify math for gas savings

### **Remediation Status**

This finding was successfully resolved using the provided recommendation.

## **4. Use msg.sender for gas savings (engn33r)**

NOTE: engn33r is unsure whether this is a gas savings, but Paul thinks it is: [https://github.com/OlympusDAO/olympus-contracts/pull/186#discussion\\_r816492541](https://github.com/OlympusDAO/olympus-contracts/pull/186#discussion_r816492541)

The emit call in the `_increaseDeposit()` function of `YieldDirector.sol` should use `msg.sender` instead of `depositInfo[depositId_].depositor`. `_addToDeposit()` is called in `_increaseDeposit()`, and `_addToDeposit()` performs a check to confirm `depositInfo[id_].depositor == msg.sender`. This is already done in the `_withdraw()` function of `YieldDirector.sol`.

### **Proof of concept**

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldDirector.sol#L404>

### **Impact**

Gas savings

### **Recommendation**

Use `msg.sender` for gas savings

### **Remediation Status**

This finding was successfully resolved using the provided recommendation.

## **5. Remove functions to save gas on deployment (engn33r)**

~~• Remove functions to save gas on deployment (ongoing)~~

The `_toAgnostic()` and `_fromAgnostic()` functions of `YieldSplitter.sol` duplicate the `balanceFrom()` and `balanceTo()` of `gOHM.sol`. Since the `gOHM` address is known to `YieldStreamer.sol` and `YieldDirector.sol`, the `_toAgnostic()` and `_fromAgnostic()` functions could be removed to reuse the `gOHM` functions. This is already done in `withdrawPrincipal()` of `YieldStreamer.sol`

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldStreamer.sol#L163>

Compare to the equivalent line in `YieldDirector.sol`

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldDirector.sol#L415>

A similar function named `getPrincipalInGOHM()` in `YieldStreamer.sol` acts as a wrapper for `gOHM.balanceTo()`. It is unclear whether this function is necessary or if it can be removed. If it is necessary, it may be useful to add a complementary function `getAgnosticInSOHM()`.

Less new code = lower complexity + deployment gas savings

## **Proof of concept**

`balanceTo` and `balanceFrom` in `gOHM.sol` convert between `sOHM` and `gOHM` using the current index exchange rate

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/governance/gOHM.sol#L111-L127>

The `YieldSplitter.sol` functions `_fromAgnostic` and `_toAgnostic` duplicate this functionality

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/types/YieldSplitter.sol#L170-L186>

## **Impact**

Gas savings

## **Recommendation**

Use existing functions instead of duplicating them

## **Remediation Status**

This finding was acknowledged but not resolved, because there may be cases where the `_toAgnostic()` and `_fromAgnostic()` functions are useful in contracts that do not import the `gOHM.sol` contract. Gas findings are optional and do not pose a security concern.

## **6. Gas savings using unchecked (engn33r)**

We know `agnosticAmount >= _toAgnostic(principal)` because `agnosticAmount` includes some yield from rebasing. This assumption can allow some math to use the "unchecked" clause for gas savings. One such place is `_getOutstandingYield` of `YieldSplitter.sol`.

## **Proof of concept**

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/types/YieldSplitter.sol#L167>

## **Impact**

Gas savings

## **Recommendation**

Use unchecked where there is no overflow or underflow risk

## **Remediation Status**

This finding was successfully resolved using the provided recommendation. It was suggested that comments explaining why each unchecked cannot overflow or underflow may be useful documentation.

## **7. Gas savings with memory variable (engn33r)**

Instead of looking up a state variable twice, a gas savings can be achieved by creating a local copy. This optimization was found in two locations.

### Proof of concept

The `_closeDeposit()` function in `YieldSplitter.sol` uses the value `"depositInfo[id_].depositor"` twice

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/types/YieldSplitter.sol#L144>

The `_redeemAll()` in `YieldDirector.sol` function uses the value `"depositInfo[receiptIds[index - 1]].depositor"` twice

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldDirector.sol#L460>

### Impact

Gas savings

### Recommendation

Store the value in a local memory variable to save gas on storage loads. Here is an example:

```
address depositorAddressToClose = depositInfo[id_].depositor;
if (depositorAddressToClose != depositorAddress) revert
YieldSplitter_NotYourDeposit();
```

### Remediation Status

This finding was successfully resolved using the provided recommendation.

## 8. redeemAll repeatedly computes index - 1 (NibblerExpress)

`index - 1` is computed six times in the `redeemAll` function. Gas could be saved by computing it once and storing in memory.

### Proof of Concept

See lines 471-480 of Yield Director: <https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldDirector.sol#L471>

## Impact

Gas savings

## Recommendation

One option is to compute and store `index - 1` and use that value rather than recomputing `index - 1` repeatedly. The alternate option would be to start the loop at `length - 1` and check that `index >= 0`.

## Remediation Status

This finding was successfully resolved using the provided recommendation.

# Informational Findings

## 1. Miscellaneous Improvement Ideas (engn33r)

There is significant overlap between `YieldDirector.sol` and `YieldStreamer.sol`, but similar functions are implemented differently. Standardizing as much as possible would be ideal for the long term management of the code.

Separately, constant variables should be used in place of magic numbers to prevent typos. The magic number `1e6` is found in multiple places in `YieldStreamer.sol` and should be replaced with a constant.

## Proof of concept

Examples of overlap between `YieldDirector.sol` and `YieldStreamer.sol` include:

- `YieldDirector.sol` has modifiers `isInvalidDeposit`, `isInvalidUpdate`, and `isInvalidWithdrawal`, but `YieldStreamer.sol` only checks the boolean value `depositDisabled`, `withdrawDisabled`, or `upkeepDisabled`. `YieldStreamer.sol` may benefit from the same modifiers used in `YieldDirector.sol`.
- Functions like `desposit()` and `addToDeposit()` perform almost identical actions but have different names for input parameters and different natspec comments. These could be standardized better across the two contracts.

Examples of magic numbers that should be converted to constant variables are found in several places:

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldStreamer.sol#L436>

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldStreamer.sol#L445>

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldStreamer.sol#L251>

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldStreamer.sol#L318>

<https://github.com/OlympusDAO/olympus-contracts/blob/f92425ff278b93cfc2a580299924417182da432a/contracts/peripheral/YieldStreamer.sol#L327>

## Impact

Informational

## Recommendation

1. Reuse code as much as possible between YieldDirector.sol and YieldStreamer.sol to reduce overall complexity
2. Replace magic numbers with constant variables to prevent typos in numeric calculations

## Remediation Status

This finding was successfully resolved using the provided recommendation.

## 2. updateUserMinDaiThreshold assumes Dai is used as the streamToken (sjkelleyjr)

### Proof of concept

<https://github.com/OlympusDAO/olympus-contracts/blob/main/contracts/peripheral/YieldStreamer.sol#L280>

## Impact

None, readability improvement.

### **Recommendation**

Rename the function to `updateUserMinStreamTokenThreshold`.

### **Remediation Status**

This finding was successfully resolved using the provided recommendation.

## **3. Missing calls in scripts/deployAll.js (engn33r)**

The `deployAll.js` script currently fails because the timelock is disabled, which leads to the calls to `olympusTreasury.queueTimeLock()` failing.

The `deployAll.js` script is missing calls to deploy `YieldStreamer.sol` and `YieldSplitter.sol`.

### **Proof of concept**

Start ganache and run `npx hardhat run --network localhost scripts/deployAll.js`. Observe the error message stating `revert Timelock is disabled, use enable`.

To fix, add the line `await olympusTreasury.initialize();` before the calls to `olympusTreasury.queueTimeLock()` and observe the above hardhat work.

### **Impact**

Informational

### **Recommendation**

Update the `deployAll.js` script

### **Remediation Status**

This finding was not resolved at the time of report completion, but does not pose any security concerns.

## **Final remarks**

## Final Comments

### **sjkelleyjr**

In general the code doesn't appear to contain any critical vulnerabilities that would result in the loss of user funds. My primary concern is around the high gas usage of the `upkeep` function and the easy with which a nefarious user could DDOS this function.

### **engn33r**

Overall the code looks quite solid. I was expecting to see a mixup between `gOHM` and `sOHM` conversion somewhere, but the math appears to be handled properly. The gas usage of the `upkeep` function, where one user pays for the gas of many user's deposits, could be problematic because the user calling `upkeep` cannot control these other deposits. Additionally, there is one comment that `streamToken` may include tokens other than `Dai`. The code currently may not handle all edge cases for deflationary or fee-on-transfer stream tokens, which could lead to problems if they are allowed as `streamTokens`.

### **NibblerExpress**

I spent a bunch of my time running through the deposit and withdraw logic looking for attacks and manipulations, but I did not find much there. The code appeared to be well written with the vulnerabilities tending to show up in the interactions with other Olympus/Ohm contracts or external contracts. A more methodical approach to testing with a focus on ensuring that all states are correct after external interactions would be helpful.

## **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