yAcademy Ohm IncurDebt review

Review Resources:
Documentation was provided

Residents:

- Peep (xiangan.eth)
- NibblerExpress

Table of Contents

tags: `Review` `yAcademy`

Review Summary

Ohm IncurDebt

The goal of the IncurDebt project is to allow other whitelisted partners to borrow ohm against their gOHM to be used to provide liquidity for tokens to be paired with OHM. Since OHM is a rebasing token when staked, there are financial disadvantages to providing liquidity for OHM since you are not capturing the supply inflation through staking. By allowing partners to borrow OHM against gOHM with 0 interest to LP, it incentivises them to hold gOHM in their treasury at the same time as increasing liquidity of their token-OHM pair.

The IncurDebt branch of the Olympus-contracts Repo was reviewed over 14 days. 2 days was used to create a report of the findings. The contracts were reviewed from May 30 to June 13. The code was reviewed by 2 resident for a total of 57 man hours (Peep 35 hours, and NibblerExpress 22 hours). 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 a936c45564692428e766727f0cae4035c02fe162. 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 Ohm team, fixes were made and included up to commit a936c45564692428e766727f0cae4035c02fe162. Fixes were reviewed to confirm whether they remedied the original finding. A security review was performed on the revised code (e.g. to determine whether new vulnerabilities were introduced by the fixes) and found no additional issues.

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, Olympus DAO 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 view functions are made public for liquidation purposes.
Mathematics	Good	Solidity ^0.8.10 is used, which provides overflow and underflow protect. Unchecked code was checked in this audit to yield no anomalies. No low-level bitwise operations are performed. There was no unusually complex math, except perhaps some of the calculations with current interest rates based on market growth factor.
Complexity	Medium	The multiple user-accessible functions and wrappings and unwrappings of OHM creates complexity in the codebase.
Libraries	Fair	SafeERC20 is used as a basic library for ERC20 transfers, but OlympusAccessControlledV2 is also used as a Olympus admin control library. Besides for that, simple interfaces are used. Fewer and simpler external dependencies is always a plus for security.
Decentralization	Fair	Currently, OlympusAccessControlledV2 is being used as the permissions modifier and governance controller.
Code stability	Average	Changes were reviewed at a specific commit and the scope was not expanded after the review was started. However, development was ongoing when the review was performed so the code was not fully frozen, which means deployed code may vary from what was reviewed.
Documentation	Good	Comments existed in many places in IncurDebt.sol, and mostly clarified what the code did. 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	Good	All tests were passing after a small fix to a typo, and 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

Critical Findings

1. Critical - Changes to index causes contract to have insufficient funds (NibblerExpress)

As background, the staking contract provides users with the ability to stake/unstake to convert between OHM and sOHM and to wrap/unwrap to convert between sOHM and gOHM. sOHM benefits from rebasing that increases the sOHM balance of users based on profits. The staking contract uses a conversion rate called the index (exposed through the gOHM `balanceTo`/`balanceFrom` functions) to change between gOHM and sOHM. The index is increased with rebasing. Importantly, OHM does not benefit from rebasing.

IncurDebt.sol allows users to deposit and withdraw gOHM, but it lends money to LPs or individual users in OHM. The contract does not charge interest or otherwise profit from lending OHM. The contract has liabilities denominated in gOHM and assets denominated in OHM. When rebasing occurs and the index changes (and OHM loans are outstanding), the liabilities increase relative to assets, and the contract is rendered insolvent. It will not have sufficient tokens to pay back the gOHM it owes.

Example 1: A user deposits 10 gOHM when the index is 100. The contract unwraps all of the gOHM (IncurDebt.sol, L467), and the user gets 1000 sOHM. The user borrows 200 OHM and leaves 800 sOHM remaining in the contract. The index increases to 200. The contract now has 1600 sOHM. The user debt (200 OHM) is now equivalent to 1 gOHM. If the user wants to withdraw the remaining 9 gOHM, the contract must wrap 1800 sOHM. The contract only has 1600 sOHM, so it cannot pay back the user in full.

Example 2: A user deposits 2 gOHM when the index is 100. The user borrows 200 OHM. The index increases to 200. The user debt is now equivalent to 1 gOHM. The user wants to withdraw the remaining 1 gOHM. The contract has no gOHM to give back to the user.

Proof of concept

In IncurDebt.sol, L459-L460 and L463-L470, borrower debt in OHM is added with first index:

```
        borrower.debt += uint128(_ohmAmount);
        totalOutstandingGlobalDebt += _ohmAmount;

        ***

        uint256 totalDebtInGOHM = IgOHM(gOHM).balanceTo(borrower.debt);

        if (totalDebtInGOHM > borrower.unwrappedGOHM) {
            uint128 amountToUnwrap = borrower.collateralInGOHM - borrower.unwrappedGOHM;
            borrower.unwrappedGOHM += amountToUnwrap;
            IStaking(staking).unwrap(address(this), amountToUnwrap);
        }
```

In L488-L491 and L496-L498, debt is paid using gOHM and the index can be different from the first:

```
        totalOutstandingGlobalDebt -= debt;
        borrower.debt = 0;
        uint256 sOHMToWrap = IgOHM(gOHM).balanceFrom(borrower.unwrappedGOHM) - debt;
        borrower.unwrappedGOHM = 0;

        ***

        IStaking(staking).wrap(address(this), sOHMToWrap);
        collateralRemaining = borrower.collateralInGOHM - IgOHM(gOHM).balanceTo(debt);
```

In L429 and L434-L435, debt is paid with OHM without accounting for the index:

```
function repayDebtWithOHM(uint256 _ohmAmount) external override isBorrower(msg.sender) {

        ***

        totalOutstandingGlobalDebt -= _ohmAmount;
        borrower.debt -= uint128(_ohmAmount);
```

Impact

Critical. The contract will not be able to repay funds nominally owed to users.

Recommendation

`Borrower.debt` should be denominated in gOHM or otherwise take account of changes in the index value.

Developer Response

When you unwrap gOHM to sOHM, even though index increases, the amount of sOHM increases too along with it. So holding sOHM is equivalent of holding gOHM. Developer acknowledged.

High Findings

1. High - Balance integer overflow revert at large balances (Peep)

In the following two examples, when _ohmAmount input is greater than type(uint128).max in these user-accessible functions, a revert will occur.

Proof of concept

When _ohmAmount is greater than type(uint128).max, L249-251 and L435-437 of IncurDebt.sol.

```
 function deposit(uint256 _amount) external override isBorrower(msg.sender) {
     borrowers[msg.sender].collateralInGOHM += uint128(_amount);

     IERC20(gOHM).safeTransferFrom(msg.sender, address(this), _amount);

     emit BorrowerDeposit(msg.sender, _amount);
 }
```

```
function repayDebtWithOHM(uint256 _ohmAmount) external override isBorrower(msg.sender) {
     Borrower storage borrower = borrowers[msg.sender];

     if (borrower.debt == 0) revert IncurDebt_BorrowerHasNoOutstandingDebt(msg.sender);

     totalOutstandingGlobalDebt -= _ohmAmount;
     borrower.debt -= uint128(_ohmAmount);

     IERC20(OHM).safeTransferFrom(msg.sender, address(this), _ohmAmount);
     ITreasury(treasury).repayDebtWithOHM(_ohmAmount);

     emit DebtPaidWithOHM(msg.sender, _ohmAmount, borrower.debt, totalOutstandingGlobalDebt);

 }
```

Impact

The uint128 typecasting should work for updating borrower.debt, but safeTransferFrom is called using the uint256 value. This is unlikely to happen, but the contract will steal a bunch of the user's money if it does.

Thankfully, Solidity 0.8.10 and above have reverts for integer underflow and overflow, so this will most likely only result in a DoS on deposits of _ohmAmount greater than type(uint128).max.

Recommendation

An easy solution can be to define _ohmAmount in `borrower.debt -= uint128(_ohmAmount);` as a uint256, the same as the input. Gas costs will be impacted minimally (if at all).

Developer Response

Acknowledged and regarded as informational because in order for max to hit they would have to deposit an absurdly large amount of gOHM.

Medium Findings

1. Medium - Typo in amount transferred from user (Peep)

A typo in Uniswap.sol causes the code to transfer `amountBDesired` rather than `amountADesired`.

Proof of concept

L77-L78 say,

```
        IERC20(tokenA).safeTransferFrom(_user, address(this), amountBDesired);
        IERC20(tokenA).approve(address(router), amountBDesired);
```

They should say,

```
        IERC20(tokenA).safeTransferFrom(_user, address(this), amountADesired);
        IERC20(tokenA).approve(address(router), amountADesired);
```

Impact

Medium. The contract may transfer too many funds from the user and be unable to return them.

Recommendation

Replace `amountBDesired` with `amountADesired` as indicated above.

Developer Response

Fixed

2. Medium - Overflow revert on depositing an amount greater than debt amount (Peep)

In the following example, when `_ohmAmount` input is greater than `borrower.debt` in the user accessible `repayDebtWithOHM()`, borrower.debt will overflow if uint128(_ohmAmount) > borrower.debt.

Proof of concept

When `_ohmAmount` is greater than `borrower.debt`, as seen in L435-437 of IncurDebt.sol,.

```
    function repayDebtWithOHM(uint256 _ohmAmount) external override isBorrower(msg.sender) {
        Borrower storage borrower = borrowers[msg.sender];

        if (borrower.debt == 0) revert IncurDebt_BorrowerHasNoOutstandingDebt(msg.sender);

        totalOutstandingGlobalDebt -= _ohmAmount;
        borrower.debt -= uint128(_ohmAmount);

        IERC20(OHM).safeTransferFrom(msg.sender, address(this), _ohmAmount);
        ITreasury(treasury).repayDebtWithOHM(_ohmAmount);

        emit DebtPaidWithOHM(msg.sender, _ohmAmount, borrower.debt, totalOutstandingGlobalDebt);

    }
```

The case with global outstanding debt is unlikely to happen (but can still happen if, for example, there is only one borrower and the `totalOutstandingGlobalDebt == borrower.debt`).

Impact

Borrower debt will overflow and cause the user a DoS due to overflowed debt.

Recommendation

Implement a revert error where a user cannot return more than their debt amount.

Developer Response

Intended behavior. user should not be able to repay more than they have in debt

Low Findings

1. Low - No check for `_amount > type(uint128).max` (NibblerExpress)

IncurDebt.sol has functions that receive `uint256` values and typecasts them to `uint128` for some operations but not others.

Proof of concept

In L249-L251 and L389-L390 of IncurDebt.sol, `_amount` is typecast to `uint128` for updating the borrower's collateral but not for transferring funds.

```
        borrowers[msg.sender].collateralInGOHM += uint128(_amount);
        IERC20(gOHM).safeTransferFrom(msg.sender, address(this), _amount);

        ***

        borrower.collateralInGOHM -= uint128(_amount);
        IERC20(gOHM).transfer(msg.sender, _amount);
```

Impact

Low. It is very unlikely a user will have more than `type(uint128).max` gOHM, but loss of funds would be possible if that happened. Withdraw has additional checks that should prevent users from transferring out more than their collateral.

Recommendation

Typecast `_amount` to uint128 for all operations or revert when `_amount > type(uint128).max`.

Developer Response

Acknowledged, again like the comment before: not a concern since uint128 is quite big. Should revert if user mistakenly inputs too big number.

2. Low - Fee on transfer tokens can get stuck (NibblerExpress)

The strategies allows users to add an arbitrary token to an LP contract. A user may deposit a fee on transfer token. The deposit may succeed if the amount received by the LP contract is more than the minimum amount. When removing the liquidity, the LP contract will report that it returned more tokens than it did. The strategy contract will attempt to return those tokens to the user and revert. The tokens will be stuck in the LP contract.

Proof of concept

Uniswap strategy adds liquidity here.

Liquidity is removed from the LP contract and returned to the user at L148-L166 of Uniswap.Sol. If the LP contract reports the wrong amount returned due to a fee on transfer, the transfer to the user will fail.

```
(uint256 amountA, uint256 amountB) = router.removeLiquidity(
            tokenA,
            tokenB,
            liquidity,
            (amountAMin * slippage) / 1000,
            (amountBMin * slippage) / 1000,
            address(this),
            block.timestamp
        );

        if (tokenA == ohmAddress) {
            ohmRecieved = amountA;
            IERC20(tokenA).safeTransfer(incurDebtAddress, amountA);
            IERC20(tokenB).safeTransfer(_user, amountB);
        } else {
            ohmRecieved = amountB;
            IERC20(tokenB).safeTransfer(incurDebtAddress, amountB);
            IERC20(tokenA).safeTransfer(_user, amountA);
        }
```

The logic in Curve.sol functions similarly.

Impact

Low. It may not be possible to withdraw an LP position without sending extra of the fee on transfer tokens to the contract.

Recommendation

Send the smaller of the balance of tokens in the contract and the amount reported as returned by the LP contract (e.g., `amountA` or `amountB`).

Developer Response

Fixed

3. Low - Reentrant token can take tokens returned to user by Balancer (NibblerExpress)

The Balancer.sol contract returns all contract tokens of each type to the user. A reentrant token could call `removeLiquidity()` from an attacker address before all the tokens have been transferred to the user and receive some of the users tokens.

Proof of concept

A reentrant token could cause the for loop at IncurDebt.sol L127-L135 to be paused before all assets are transferred to the user and run again for the attacker's address. The tokens intended for the user would be transferred to the attacker.

```
        for (uint256 i = 0; i < assets.length; i++) {
            if (assets[i] == ohmAddress) {
                ohmRecieved = IERC20(ohmAddress).balanceOf(address(this));
                IERC20(ohmAddress).safeTransfer(incurDebtAddress, ohmRecieved);
            } else {
                uint256 balance = IERC20(assets[i]).balanceOf(address(this));
                IERC20(assets[i]).safeTransfer(_user, balance);
            }
        }
```

Impact

Low. Theft of funds is possible, but it would require the user to include the attacker's token as one of the tokens added to the balancer pool.

Recommendation

Make `removeLiquidity()` non-reentrant or calculate the correct balance to send the user.

Developer Response

Fixed

Gas Savings Findings

1. Gas Savings - Duplicative checks of isBorrower (NibblerExpress)

In IncurDebt.sol, `seize` checks `isBorrower` and immediate calls `_repay`, which also checks `isBorrower`.

Proof of concept

`seize` check:

```
    function seize(address _borrower) external override onlyGovernor isBorrower(_borrower) {
        (uint256 seizedCollateral, uint256 paidDebt) = _repay(_borrower);
```

`_repay` check:

```
    function _repay(address _borrower)
        internal
        isBorrower(_borrower)
        returns (uint256 collateralRemaining, uint256 paidDebt)
    {
```

Impact

Gas savings.

Recommendation

Remove check of `isBorrower` from `seize`. (Note that `forceRepay`, `repayDebtWithCollateral`, and `repayDebtWithCollateralAndWithdrawTheRest` all rely on `_repay` to check `isBorrower`.)

Developer Response

Fixed

2. Gas Savings - Rely on checked math (NibblerExpress)

In IncurDebt.sol, `removeLP` checks whether the user is taking out too much liquidity and then immediately does a math calculation that will revert if the user is taking out too much liquidity.

Proof of concept

L324-L327 say,

```
        if (_liquidity > lpTokenOwnership[_lpToken][msg.sender])
            revert IncurDebt_AmountAboveBorrowerBalance(_liquidity);

        lpTokenOwnership[_lpToken][msg.sender] -= _liquidity;
```

L359-L364 say,

```
        if (_liquidity > lpTokenOwnership[_lpToken][msg.sender])
            revert IncurDebt_AmountAboveBorrowerBalance(_liquidity);

        if (borrowers[msg.sender].debt != 0) repayDebtWithCollateral();

        lpTokenOwnership[_lpToken][msg.sender] -= _liquidity;
```

Impact

Gas savings.

Recommendation

You could leave as is if you want the custom error message. Otherwise, the if statement and revert can be removed.

Developer Response

Fixed

3. Gas Savings - Don't check `isBorrower` or `isLPBorrower` on withdraw (NibblerExpress)

Non-borrowers should have a balance of zero, so there is no need to check whether someone is a borrower on a withdraw or repay function.

Proof of concept

L321-L327 say,

```
        Borrower storage borrower = borrowers[msg.sender];
        if (!borrower.isLpBorrower) revert IncurDebt_NotBorrower(msg.sender);

        if (_liquidity > lpTokenOwnership[_lpToken][msg.sender])
            revert IncurDebt_AmountAboveBorrowerBalance(_liquidity);

        lpTokenOwnership[_lpToken][msg.sender] -= _liquidity;
```

`lpTokenOwnership[_lpToken][msg.sender]` should be zero for non-borrowers.

This is also true for:
`withdrawLP` - L356
`withdraw` - L374
`repayDebtWithOHM` - L429
`_repay` - L479

Impact

Gas savings.

Recommendation

You could leave as is if you want the custom error message. Otherwise, remove the `isBorrower` and `isLpBorrower` checks.

Developer Response

Acknowledged, unchanged as an extra layer of security.

Informational Findings

1. SPDX license identifiers not provided in source files of IncurDebt.sol and other similar library contracts (IStrategy.sol, etc.) (Peep)

Proof of concept

Compiling the contracts produce such a warning.

Impact

Please see https://spdx.org for more information.

Recommendation

Consider adding in license such as

```
// SPDX-License-Identifier: MIT
```

Developer Response

2. Addresses yet to be hardcoded (no zero checks) on constructor (Peep)

Proof of concept

Addresses in IncurDebt.sol L102-L114 are not hardcoded, without zero checks.

```
    constructor(
        address _OHM,
        address _gOHM,
        address _sOHM,
        address _staking,
        address _treasury,
        address _olympusAuthority
    ) OlympusAccessControlledV2(IOlympusAuthority(_olympusAuthority)) {
        OHM = _OHM;
        gOHM = _gOHM;
        sOHM = _sOHM;
        staking = _staking;
        treasury = _treasury;
```

This is just a warning that error is possible on deployment.

Impact

Incorrect deployment may lead to gas burn / necessity for redeploy.

Recommendation

Consider hardcoding addresses as seen here (easier), or implementing zero checks on addresses to control for deploy errors.

Developer Response

Fixed

3. Informational - Typos (Peep, NibblerExpress)

(Persistent) Typo on revert error for input amount 0 of `withdraw`. Many occurrences of `OhmRecieved` - should be `OhmReceived`. Comment includes "debit" instead of "debt."

Proof of concept

L375 in IncurDebt.sol says,

```
if (_amount == 0) revert IncurDebt_InvaildNumber(_amount);
```

should be

```
if (_amount == 0) revert IncurDebt_InvalidNumber(_amount);
```

L314-L344 use `ohmRecieved`. For example,

```
    function removeLP(
        uint256 _liquidity,
        address _strategy,
        address _lpToken,
        bytes calldata _strategyParams
    ) external isStrategyApproved(_strategy) returns (uint256 ohmRecieved) {
```

L475-L477 use "debit" when they appear to mean "debt."

```
    /// @notice pays borrower debit and return left collateral and paid debt
    /// @param _borrower the account debit needs to be paid
    /// @return collateralRemaining left collateral after paying debit
```

Impact

May cause additional edits in the future with correct spelling to cause errors.

Recommendation

Fix spelling errors across all occurrences.

Developer Response

Fixed

4. No pragma declaration on ICurvePoolFactory.sol (Peep)

Proof of concept

```
interface ICurvePoolFactory {
```

starts with no pragma declaration

Impact

Leads to revert on deployment

Recommendation

Add compiler version via a pragma declaration

Developer Response

5. Informational - Strategies must check `_lpToken` (NibblerExpress)

IncurDebt.sol does not check whether the `_lpToken` matches `_strategy` and `_strategyParams`. The strategies are relied on to do this check. If not checked, users can pass `_lpToken` for one pool and remove liquidity for another.

Proof of concept

In IncurDebt.sol L324-L325, The liquidity check will be wrong if the user passes a value for `_lpToken` that doesn't match the actual pool they are removing funds from.

```
        if (_liquidity > lpTokenOwnership[_lpToken][msg.sender])
            revert IncurDebt_AmountAboveBorrowerBalance(_liquidity);
```

Impact

Informational. The reviewed strategies either check that `_lpToken` matches the actual information from `_strategyParams` or use `_lpToken` directly to access the pool.

Recommendation

Ensure all future whitelisted strategies are using or checking `_lpToken`.

Developer Response

Intentional design

6. Informational - Comments don't match code (NibblerExpress, Peep)

Some comments include requirements that are not checked in the code.

Proof of concept

L137 - The comment says, "- must be greater than or equal to existing debt." This is not checked.

```
    /// @notice sets the maximum debt limit for the system
    /// - onlyOwner (or governance)
    /// - must be greater than or equal to existing debt
    /// @param _limit in OHM
    function setGlobalDebtLimit(uint256 _limit) external override onlyGovernor {
        globalDebtLimit = _limit;
        emit GlobalLimitChanged(_limit);
    }
```

IncurDebt.sol L166 - The comment says "- user must not be borrower" but the function only checks if the user is already an LP borrower, not both types of borrower.

```
    function allowNonLPBorrower(address _borrower) external override onlyGovernor {
        if (borrowers[_borrower].isLpBorrower) revert IncurDebt_AlreadyBorrower(_borrower); //@audit add addit
revert for nonlpborrowers?

        borrowers[_borrower].isNonLpBorrower = true;

        emit BorrowerAllowed(_borrower, false, true);
    }
```

IncurDebt.sol L182 - The comment says, "- limit must be less than or equal to the global debt limit." This is not checked, but something similar is checked at L461.

```
    /// @notice sets the maximum debt limit for a borrower
    /// - onlyOwner (or governance)
    /// - limit must be greater than or equal to borrower's outstanding debt
    /// - limit must be less than or equal to the global debt limit
    /// @param _borrower the address that will interact with contract
    /// @param _limit borrower's debt limit in OHM
    function setBorrowerDebtLimit(address _borrower, uint256 _limit)
        external
        override
        onlyGovernor
        isBorrower(_borrower)
    {
        if (_limit < borrowers[_borrower].debt) revert IncurDebt_AboveBorrowersDebtLimit(_limit);

        borrowers[_borrower].limit = uint128(_limit);
        emit BorrowerDebtLimitSet(_borrower, _limit);
    }
```

Impact

Informational. The comments are inconsistent with the code, but the code should still work properly.

Recommendation

Revise the comments or the code for consistency.

Developer Response

Fixed

7. Informational - Unwrap only required amount (NibblerExpress)

In IncurDebt.sol, `_borrow` unwraps all wrapped GOHM rather than just what is needed.

Proof of concept

L465-L467 of IncurDebt.sol say,

```
        if (totalDebtInGOHM > borrower.unwrappedGOHM) {
            uint128 amountToUnwrap = borrower.collateralInGOHM - borrower.unwrappedGOHM;
```

They could instead say,

```
        if (totalDebtInGOHM > borrower.unwrappedGOHM) {
            uint128 amountToUnwrap = totalDebtInGOHM - borrower.unwrappedGOHM;
```

Impact

Informational. In some cases, the contract can avoid wrapping sOHM during withdrawal by not unwrapping unneeded amounts during borrowing.

Recommendation

Revise the code as proposed.

Developer Response

Fixed

8. Informational - Replace magic numbers (NibblerExpress)

Uniswap.sol and Balancer.sol use magic numbers. The numbers should be replaced with constants to prevent errors and add a description of what the number does.

Proof of concept

Examples include the number 1000 in L86-L95 and L148-L156 of Uniswap.sol

```
        (amountA, amountB, liquidity) = router.addLiquidity(
            tokenA,
            tokenB,
            amountADesired,
            amountBDesired,
            (amountAMin * slippage) / 1000,
            (amountBMin * slippage) / 1000,
            incurDebtAddress,
            block.timestamp
        );

        ***

        (uint256 amountA, uint256 amountB) = router.removeLiquidity(
            tokenA,
            tokenB,
            liquidity,
            (amountAMin * slippage) / 1000,
            (amountBMin * slippage) / 1000,
            address(this),
            block.timestamp
        );
```

Additional examples including "1" for the join and exit kinds at L80 and L113 of Balancer.sol.

```
        bytes memory userData = abi.encode(1, maxAmountsIn, minimumBPT);

        ***

        bytes memory userData = abi.encode(1, _liquidity);
```

Impact

Informational. Changes will improve readability and prevent errors in forks or future revisions.

Recommendation

Replace magic numbers with constants.

Developer Response

Fixed

9. Informational - Confusing use of slippage (NibblerExpress)

Uniswap.sol receives a minimum amount from the user and a slippage. It then multiplies them together, so the actual minimum is less than the nominal minimum by the slippage amount. This may be confusing for users depending on how this is exposed to them.

Proof of concept

Examples include the `amountAMin` and `amountBMin` by `slippage` in L86-L95 and L148-L156 of Uniswap.sol

```
        (amountA, amountB, liquidity) = router.addLiquidity(
            tokenA,
            tokenB,
            amountADesired,
            amountBDesired,
            (amountAMin * slippage) / 1000,
            (amountBMin * slippage) / 1000,
            incurDebtAddress,
            block.timestamp
        );

        ***
```

```
        (uint256 amountA, uint256 amountB) = router.removeLiquidity(
            tokenA,
            tokenB,
            liquidity,
            (amountAMin * slippage) / 1000,
            (amountBMin * slippage) / 1000,
            address(this),
            block.timestamp
        );
```

Impact

Informational. Changes will make it more likely the front end or future contracts will be implemented correctly.

Recommendation

Multiply `amountADesired` and `amountBDesired` by `slippage` or just compute `amountAMin` and `amountBMin` on the front end based on the slippage.

Developer Response

Fixed

10. Informational - No sweep function (NibblerExpress)

IncurDebt.sol and the strategies do not have any sweep functions, so additional rewards or air drops cannot be removed from the contracts.

Proof of concept

The following contracts do not have sweep functions:
IncurDebt.sol
UniSwap.sol
Curve.sol
Balancer.sol

Impact

Informational. Unexpected rewards and air drops cannot be removed.

Recommendation

Include a sweep function in each contract, and have a blacklist that prohibits sweeping of OHM, sOHM, gOHM, all lpTokens added, and all tokens received from users for deposit into a vault.

Developer Response

Fixed

Final remarks

NibblerExpress: The contract was well written. There were few logical errors. The accounting will need to be corrected to ensure that liabilities do not grow beyond assets. The strategies take separate parameters from IncurDebt.sol and transfer arbitrary tokens. The initial strategies present a large attack surface but seem mostly safe. Future strategies should be carefully reviewed to ensure attacks are not possible.

Peep: Very well written contract with seemingly little issues. A patch regarding fixing most of these findings was published shortly after publishing the first draft of our findings, in which we did not find any additional vulnerabilities upon review.

Appendix and FAQ

tags: `Review` `yAcademy`