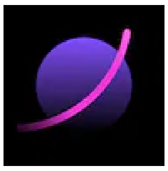

Mellow Protocol contest Findings & Analysis Report

2022-02-09

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 code contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the code contest outlined in this document, C4 conducted an analysis of Mellow Protocol contest smart contract system written in Solidity. The code contest took place between December 2—December 8 2021.

Wardens

17 Wardens contributed reports to the Mellow Protocol contest:

1. WatchPug (jtp and ming)
2. cmichel
3. gzeon
4. 0x1f8b
5. 0x0x0x

6. MetaOxNull
7. robee
8. pauliax
9. cuong_qnom
10. hagrid
11. 0x421f
12. defsec
13. Jujic
14. yeOlde
15. pmerkleplant
16. hyh

This contest was judged by 0xleastwood.

Final report assembled by itsmetechjay and CloudEllie.

Summary

The C4 analysis yielded an aggregated total of 23 unique vulnerabilities and 73 total findings. All of the issues presented here are linked back to their original finding.

Of these vulnerabilities, 4 received a risk rating in the category of HIGH severity, 7 received a risk rating in the category of MEDIUM severity, and 12 received a risk rating in the category of LOW severity.

C4 analysis also identified 11 non-critical recommendations and 39 gas optimizations.

Scope

The code under review can be found within the C4 Mellow Protocol contest repository, and is composed of 80 smart contracts written in the Solidity programming language and includes 5400 lines of Solidity code and 3 lines of JavaScript.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (4)

[H-01] `YearnVault.sol#pull()` will most certainly fail

Submitted by WatchPug

https://github.com/code-423n4/2021-12-mellow/blob/6679e2dd118b33481ee81ad013ece4ea723327b5/mellow-vaults/test_brownie/contracts/YearnVault.sol#L84-L101

```
        for (uint256 i = 0; i < _yTokens.length; i++) {
            if (tokenAmounts[i] == 0) {
                continue;
            }

            IYearnVault yToken = IYearnVault(_yTokens[i]);
            uint256 yTokenAmount = ((tokenAmounts[i] * (10**yToken.c
            uint256 balance = yToken.balanceOf(address(this));
            if (yTokenAmount > balance) {
                yTokenAmount = balance;
            }
            if (yTokenAmount == 0) {
                continue;
            }
            yToken.withdraw(yTokenAmount, to, maxLoss);
            (tokenAmounts[i], address(this));
        }
        actualTokenAmounts = tokenAmounts;
```

The actual token withdrew from `yToken.withdraw()` will most certainly be less than the `tokenAmounts[i]`, due to precision loss in the calculation of `yTokenAmount`.

As a result, `IERC20(_vaultTokens[i]).safeTransfer(to, actualTokenAmounts[i]);` in `LpIssuer.sol#withdraw()` will revert due to insufficant balance.

Recommendation

Change to:

```
    tokenAmounts[i] = yToken.withdraw(yTokenAmount, to, maxLoss);
```

[MihanixA (Mellow Protocol) confirmed and disagreed with severity]:

> Actually I don't see how this could lead to fund loss. I think this one is a bug. @0xleastwood what do you think?

[0xleastwood (judge) commented]:

> my understanding is that users won't be able to withdraw pushed funds @MihanixA

> so fund loss is related to not being able to withdraw rather than by extracting value from the protocol

[0xn1ck (Mellow Protocol) commented]:

> While we agree that this will prevent full withdrawal of the funds, that wil be limited to only a couple of wei's which is the yearn precision loss. So in case you put 100eth you will be able to recover 100eth - 1wei. So we'd rather name the issue "some small amounts cannot be withdrawn from the pool"

[0xleastwood (judge) commented]:

> If my understanding is correct, `YearnVault._pull` will withdraw `yTokenAmount` representing the yToken's shares and then withdraw on this amount but return `tokenAmounts` where the amount withdrawn is typically less than the amount intended to be withdrawn. `LpIssuer.withdraw()` will expect `actualTokenAmounts` to be available to be transferred which isn't exactly in the contract's balance.

- https://github.com/code-423n4/2021-12-mellow/blob/6679e2dd118b33481ee81ad013ece4ea723327b5/mellow-vaults/contracts/YearnVault.sol#L90
- https://github.com/code-423n4/2021-12-mellow/blob/6679e2dd118b33481ee81ad013ece4ea723327b5/mellow-vaults/contracts/LpIssuer.sol#L152

> Let's use an example:

- Alice calls `LpIssuer.withdraw()` with `tokensAmount[0]` equal to 100 tokens. Let's ignore the `lpTokenAmount` argument for the sake of this example.
- `_subvault().pull` is called on this `tokensAmount[0]`.
- `yTokenAmount` is calculated according to `((tokenAmounts[i] * (10**yToken.decimals())) / yToken.pricePerShare());` which potentially leads to a slightly truncated output.
- This truncated output represents the shares belonging to the user which is then parsed to `yToken.withdraw()`.
- `yToken.withdraw()` is likely less than 100 tokens and is sent to the `LpIssuer.sol` contract but `actualTokenAmounts[0]` is equal to 100 tokens.
- `LpIssuer.withdraw()` attempts to send tokens to the withdrawer but is unable as the contract does not have sufficient balance.

```
    IERC20(_vaultTokens[i]).safeTransfer(to, actualTokenAmounts[i]);
```

- If I'm not mistaken, it seems like this issue would be apparent on any withdrawal amount (assuming there is some amount truncated).
- There is also an important edge case where the amount to withdraw from the yearn vault is greater than the available contract balance, it will always revert.

Oxn1ck (Mellow Protocol) commented:

> Agreed, thank you!

[H-02] Wrong implementation of `performanceFee` can cause users to lose 50% to 100% of their funds

Submitted by WatchPug

A certain amount of lp tokens (shares of the vault) will be minted to the `strategyPerformanceTreasury` as `performanceFee`, the amount is calculated based on the `minLpPriceFactor`.

However, the current formula for `toMint` is wrong, which issues more than 100% of the current totalSupply of the lp token to the `strategyPerformanceTreasury` each time. Causing users to lose 50% to 100% of their funds after a few times.

https://github.com/code-423n4/2021-12-mellow/blob/6679e2dd118b33481ee81ad013ece4ea723327b5/mellow-vaults/contracts/LpIssuer.sol#L269-L271

```
    address treasury = strategyParams.strategyPerformanceTreasury;
    uint256 toMint = (baseSupply * minLpPriceFactor) / CommonLibrary
    _mint(treasury, toMint);
```

Proof of Concept

Given:

- `strategyParams.performanceFee`: `10e7` (1%)

- Alice deposited `1,000 USDC`, received `1000` lpToken; the totalSupply of the lpToken is now: `1000`;
- 3 days later, `baseTvl` increased to `1,001 USDC`, Bob deposited `1 USDC` and trigegred `_chargeFees()`:
- Expected Result: `strategyPerformanceTreasury` to receive about `0.01` lpToken (1% of 1 USDC);
- Actual Result: `minLpPriceFactor` is about `1.001`, and `strategyPerformanceTreasury` will received `1001` lpToken as performanceFee; Alice lose 50% of deposited funds.

Recommendation

Change to:

```
address treasury = strategyParams.strategyPerformanceTreasury;
uint256 toMint = (baseSupply * (minLpPriceFactor - CommonLibrary
_mint(treasury, toMint);
```

MihanixA (Mellow Protocol) confirmed

[H-03] `UniV3Vault.sol#collectEarnings()` can be front run

Submitted by WatchPug

For `UniV3Vault`, it seems that lp fees are collected through `collectEarnings()` callable by the `strategy` and reinvested (rebalanced).

However, in the current implementation, unharvested yields are not included in `tvl()`, making it vulnerable to front-run attacks that steal pending yields.

- https://github.com/code-423n4/2021-12-mellow/blob/6679e2dd118b33481ee81ad013ece4ea723327b5/mellow-vaults/contracts/UniV3Vault.sol#L100-L122
- https://github.com/code-423n4/2021-12-mellow/blob/6679e2dd118b33481ee81ad013ece4ea723327b5/mellow-

vaults/contracts/UniV3Vault.sol#L80-L97

Proof Of Concept

Given:

- Current `tvl()` is `10 ETH` and `40,000 USDC`;
- Current unclaimed yields (trading fees) is `1 ETH` and `4,000 USDC`;
- `strategy` calls `collectEarnings()` to collect fees and reinvest;
- The attacker sends a deposit tx with a higher gas price to deposit `10 ETH` and `40,000 USDC`, take 50% share of the pool;
- After the transaction in step 1 is packed, the attacker calls `withdraw()` and retrieves `10.5 ETH` and `42,000 USDC`.

As a result, the attacker has stolen half of the pending yields in about 1 block of time.

Recommendation

Consider including fees in `tvl()`.

For the code to calculate fees earned, please reference `_computeFeesEarned()` in G-UNI project:

- https://github.com/gelatodigital/g-uni-v1-core/blob/master/contracts/GUniPool.sol#L762-L806

MihanixA (Mellow Protocol) confirmed:

> Thanks! Added `tokensOwed` to `UniV3Vault`'s `tvl`

[H-04] AaveVault does not update TVL on deposit/withdraw

Submitted by cmichel, also found by WatchPug

Aave uses **rebasing** tokens which means the token balance `aToken.balanceOf(this)` increases over time with the accrued interest.

The `AaveVault.tvl` uses a cached value that needs to be updated using a `updateTvls` call.

This call is not done when depositing tokens which allows an attacker to deposit tokens, get a fair share *of the old tvl*, update the tvl to include the interest, and then withdraw the LP tokens receiving a larger share of the *new tvl*, receiving back their initial deposit + the share of the interest. This can be done risk-free in a single transaction.

Proof Of Concept

- Imagine an Aave Vault with a single vault token, and current TVL = `1,000 aTokens`
- Attacker calls `LPIssuer.push([1000])` . This loads the old, cached `tvl` . No `updateTvl` is called.
- The `1000` underlying tokens are already balanced as there's only one aToken, then the entire amount is pushed: `aaveVault.transferAndPush([1000])` . This deposists `1000` underlying tokens to the Aave lending pool and returns `actualTokenAmounts = [1000]` . **After that** the internal `_tvls` variable is updated with the latest aTokens. This includes the 1000 aTokens just deposited **but also the new rebased aToken amounts**, the interest the vault received from supplying the tokens since last `updateTvls` call. `_tvls = _tvls + interest + 1000`
- The LP amount to mint `amountToMint` is still calculated on the old cached `tvl` memory variable, i.e., attacker receives `amount / oldTvl = 1000/1000 = 100%` of existing LP supply
- Attacker withdraws the LP tokens for 50% of the new TVL (it has been updated in `deposit` 's `transferAndPush` call). Attacker receives `50% * _newTvl = 50% * (2,000 + interest) = 1000 + 0.5 * interest`.
- Attacker makes a profit of `0.5 * interest`

Impact

The interest since the last TVL storage update can be stolen as Aave uses rebasing tokens but the tvl is not first recomputed when depositing. If the vaults experience low activity a significant amount of interest can accrue which can all be captured by taking a flashloan and depositing and withdrawing a large amount to capture a large share of this interest

Recommended Mitigation Steps

Update the tvl when depositing and withdrawing before doing anything else.

[MihanixA (Mellow Protocol) confirmed]

Medium Risk Findings (7)

[M-01] User deposits don't have min. return checks

Submitted by cmichel

The `LPIssuer.deposit` first computes *balanced amounts* on the user's defined `tokenAmounts`. The idea is that LP tokens give the same percentage share of each vault tokens' tvl, therefore the provided amounts should be *balanced*, meaning, the `depositAmount / tvl` ratio should be equal for all vault tokens.

But the strategist can frontrun the user's deposit and rebalance the vault tokens, changing the tvl for each vault token which changes the rebalance. This frontrun can happen accidentally whenever the strategist rebalances

Proof Of Concept

There's a vault with two tokens A and B, tvls are `[500, 1500]`

- The user provides `[500, 1500]`, expecting to get 50% of the share supply (is minted 100% of old total supply).
- The strategist rebalances to `[1000, 1000]`
- The user's balanceFactor is `min(500/1000, 1500/1000) = 1/2`, their balancedAmounts are thus `tvl * balanceFactor = [500, 500]`, the `1000` excess token B are refunded. In the end, they only received `500/(1000+500) = 33.3%` of the total supply but used up all of their token A which they might have wanted to hold on to if they had known they'd only get 33.3% of the supply.

Impact

Users can get rekt when depositing as the received LP amount is unpredictable and lead to a trade using a very different balanced token mix that they never intended.

Recommended Mitigation Steps

Add minimum return amount checks. Accept a function parameter that can be chosen by the user indicating their *expected LP amount* for their deposit `tokenAmounts`, then check that the actually minted LP token amount is above this parameter.

[MihanixA (Mellow Protocol) confirmed]()

[M-02] Withdraw from `AaveVault` will receive less than actual share

Submitted by gzeon

Impact

`AaveVault` cache `tvl` and update it at the end of each `_push` and `_pull`. When withdrawing from `LpIssuer`, `tokenAmounts` is calculated using the cached `tvl` to be pulled from `AaveVault`. This will lead to user missing out their share of the accrued interest / donations to Aave since the last `updateTvls`.

Proof of Concept

- https://github.com/code-423n4/2021-12-mellow/blob/6679e2dd118b33481ee81ad013ece4ea723327b5/mellow-vaults/contracts/LpIssuer.sol#L150
- https://github.com/code-423n4/2021-12-mellow/blob/6679e2dd118b33481ee81ad013ece4ea723327b5/mellow-vaults/contracts/AaveVault.sol#L13

Recommended Mitigation Steps

Call `updateTvls` at the beginning of `withdraw` function if the `_subvault` will cache tvl

MihanixA (Mellow Protocol) confirmed

[M-03] Users can avoid paying vault fees

Submitted by cmichel

The `LPIssuer.deposit/withdraw` function charges protocol&management&performance fees through inflating the LP supply in the `_chargeFees` function. However, this LP fees minting is skipped if the elapsed time is less than the `managementFeeChargeDelay`:

```
    if (elapsed < vg.delayedProtocolParams().managementFeeChargeDela
        return;
    }
```

This allows a user to avoid paying any fees if they deposit right after a charge fee interaction and withdraw within again `managementFeeChargeDelay` time period.

Proof Of Concept

This can be abused heavily on networks where the gas fees are a lot cheaper than the three vault fees:

- deposit a tiny amount just to trigger the charge fees. This sets `lastFeeCharge`
- deposit a huge amount, tvl increases significantly
- let it earn interest. withdraw it before the `managementFeeChargeDelay`. No fees are paid, tvl reduces significantly
- repeat, fees are only paid on tiny tvl

Impact

In the worst case, nobody pays fees by repeating the above actions.

Recommended Mitigation Steps

Fees must always be charged on each deposit and withdrawal, even within the same block as it could be that a huge interest “harvest” comes in that an attacker sandwiches. Remove the `if (elapsed < vg.delayedProtocolParams().managementFeeChargeDelay) { return; }` code.

[MihanixA (Mellow Protocol) disputed](#):

> Charging fees on every deposit costs a lot of gas. If we notice users avoid paying fees we would just switch `managementFeeChargeDelay` to zero.

[0xleastwood (judge) commented](#):

> I don't think this is a valid reason. At the end of the day, users can still abuse this feature and it is best to think about worse case scenario. I'll keep this issue as `high` severity unless there is any reason not to? @MihanixA

[MihanixA (Mellow Protocol) commented](#):

> @0xleastwood Notice that users funds are not at risk and we can not remove this behaviour because this would lead to high withdraw/deposit fees. Anyway feel free to mark this issue anything seems fair to you.

[0xleastwood (judge) commented](#):

> While I understand funds aren't directly at risk. It does seem like this issue can be exploited to earn additional yield at a lower opportunity cost as the user does not pay fees.
>
> I think I'll mark this as `medium` then.

```
2 — Med (M): vulns have a risk of 2 and are considered "Medium"
```

[M-04] `ChiefTrader.sol` Wrong implementation of `swapExactInput()` and `swapExactOutput()`

Submitted by WatchPug, also found by Meta0xNull

When a caller calls `ChiefTrader.sol#swapExactInput()` , it will call `ITrader(traderAddress).swapExactInput()` .

https://github.com/code-423n4/2021-12-mellow/blob/6679e2dd118b33481ee81ad013ece4ea723327b5/mellow-vaults/contracts/trader/ChiefTrader.sol#L59-L59

```
    return ITrader(traderAddress).swapExactInput(0, amount, recipier
```

However, in the current implementation, `inputToken` is not approved to the `traderAddress` .

For example, in `UniV3Trader.sol#_swapExactInputSingle` , at L89, it tries to transfer inputToken from `msg.sender` (which is `ChiefTrader`), since it's not approved, this will revert.

Plus, the `inputToken` should also be transferred from the caller before calling the subtrader.

https://github.com/code-423n4/2021-12-mellow/blob/6679e2dd118b33481ee81ad013ece4ea723327b5/mellow-vaults/contracts/trader/UniV3Trader.sol#L89-L89

```
    IERC20(input).safeTransferFrom(msg.sender, address(this), amount
```

The same problem exists in `swapExactOutput()` :

https://github.com/code-423n4/2021-12-mellow/blob/6679e2dd118b33481ee81ad013ece4ea723327b5/mellow-vaults/contracts/trader/ChiefTrader.sol#L63-L75

```
    function swapExactOutput(
        uint256 traderId,
        uint256 amount,
        address,
        PathItem[] calldata path,
        bytes calldata options
    ) external returns (uint256) {
        require(traderId < _traders.length, TraderExceptionsLibrary.
        _requireAllowedTokens(path);
        address traderAddress = _traders[traderId];
        address recipient = msg.sender;
        return ITrader(traderAddress).swapExactOutput(0, amount, rec
    }
```

Recommendation

Approve the `inputToken` to the subtrader and transfer from the caller before calling `ITrader.swapExactInput()` and `ITrader.swapExactOutput()`.

Or maybe just remove support of `swapExactInput()` and `swapExactOutput()` in `ChiefTrader`.

[MihanixA (Mellow Protocol) confirmed](#):

> In fact, tokens are approved to the trader https://github.com/code-423n4/2021-12-mellow/blob/6679e2dd118b33481ee81ad013ece4ea723327b5/mellow-vaults/contracts/ERC20Vault.sol#L67 The problem is that `safeTransferFrom` uses `msg.sender` which is `ChiefTrader` instead of `recipient` which is `ERC20Vault`.

[M-05] Admin can break `_numberOfValidTokens`

Submitted by cmichel, also found by gzeon and 0x1f8b

The `ProtocolGovernance._numberOfValidTokens` can be decreased by the admin in the `ProtocolGovernance.removeFromTokenWhitelist` function:

```
    function removeFromTokenWhitelist(address addr) external {
        require(isAdmin(msg.sender), "ADM");
        _tokensAllowed[addr] = false;
        if (_tokenEverAdded[addr]) {
            // @audit admin can repeatedly call this function and se
            --_numberOfValidTokens;
        }
    }
```

This function can be called repeatedly until the `_numberOfValidTokens` is zero.

Impact

The `_numberOfValidTokens` is wrong and with it the `tokenWhitelist()`.

Recommended Mitigation Steps

It seems that `_numberOfValidTokens` should only be decreased if the token was previously allowed:

```
    function removeFromTokenWhitelist(address addr) external {
        require(isAdmin(msg.sender), "ADM");
        if (_tokensAllowed[addr]) {
            _tokensAllowed[addr] = false;
            --_numberOfValidTokens;
        }
    }
```

[0xleastwood (judge) commented]():

> Can you confirm if this issue is valid or not? @MihanixA

[0xleastwood (judge) commented]():

> Just realised this is a duplicate of another issue. Marking this as the primary issue

[MihanixA (Mellow Protocol) commented]():

> @0xleastwood Confirmed, it's a bug.

[M-06] UniswapV3's path issue for `swapExactOutput`

Submitted by cmichel

UniswapV3 expects a path object like `(tokenA, feeAB, tokenB, feeBC, tokenC)`. The `UniV3Trader.swapExactOutput` code tries to reverse this path to get to `(tokenC, feeBC, tokenB, feeAB, tokenA)` but that's not what the `_reverseBytes` function does. Note that it reverts the entire encoded `path` byte array **byte-by-byte** which breaks the byte-order in a token. For example, `tokenA` would have every single byte reversed and lead to a different token.

```
function _reverseBytes(bytes memory input) internal pure returns
    /** @audit reverses byte order? */
    for (uint256 i = 0; i < input.length; ++i) output[i] = input
}
```

Impact

The `UniV3Trader.swapExactOutput` function with multi-hops is broken and cannot be used.

Recommended Mitigation Steps

Don't reverse the path byte-by-byte but element-by-element.

0xleastwood (judge) commented:

> Can you confirm if this issue is valid or not? @MihanixA

0xleastwood (judge) commented:

> I actually agree with the warden's finding here. Leaving as is.

[M-07] Bad redirects can make it impossible to deposit & withdraw

Submitted by cmichel

The `GatewayVault._push()` function gets `redirects` from the `strategyParams`. If `redirects[i] = j`, vault index `i`'s deposits are redirected to vault index `j`.

Note that the deposits for vault index `i` are cleared, as they are redirected:

```
    for (uint256 j = 0; j < _vaultTokens.length; j++) {
        uint256 vaultIndex = _subvaultNftsIndex[strategyParams.redi
        amountsByVault[vaultIndex][j] += amountsByVault[i][j];
        amountsByVault[i][j] = 0;
    }
```

> The same is true for withdrawals in the `_pull` function. Users might not be able to withdraw this way.

If the `redirects` array is misconfigured, it's possible that all `amountsByVault` are set to zero. For example, if `0` redirects to `1` and `1` redirects to `0`. Or `0` redirects to itself, etc. There are many misconfigurations that can lead to not being able to deposit to the pool anymore.

Recommended Mitigation Steps

The `redirects[i] = j` matrix needs to be restricted. If `i` is redirected to `j`, `j` may not redirect itself. Check for this when setting the `redirects` array.

0xleastwood (judge) commented:

> Can you confirm if this issue is valid or not? @MihanixA

MihanixA (Mellow Protocol) confirmed:

> @0xleastwood Confirmed

MihanixA (Mellow Protocol) commented:

> (notice that this one is a deploy-related issue)

Low Risk Findings (12)

- [L-01] `UniV3Vault` does not distribute fee earning to depositor *Submitted by gzeon*

- [L-02] Wrong logic in UniV3Trader *Submitted by 0x1f8b*
- [L-03] Unsafe token transfer *Submitted by WatchPug, also found by defsec, cuong*qnom, 0x0x0x, cmichel, and pmerkleplant_
- [L-04] Learn from the past *Submitted by 0x1f8b*
- [L-05] Potential DOS with Division By Zero on `LpIssuer` *Submitted by hagrid*
- [L-06] `maxTokensPerVault` is not used *Submitted by 0x0x0x*
- [L-07] The Contract Should Approve(0) first *Submitted by defsec, also found by Jujic and robee*
- [L-08] `AaveVault.sol#_pull()` may return wrong `actualTokenAmounts` *Submitted by WatchPug*
- [L-09] Wrong logic in `tokenWhitelist()` ? *Submitted by cmichel*
- [L-10] Guard for initialization function of VaultGovernance *Submitted by cuong*qnom_
- [L-11] `YearnVault` did not cache tvl as comment described *Submitted by gzeon*
- [L-12] adminApprove will not work *Submitted by pauliax*

Non-Critical Findings (11)

- [N-01] What you guys mean by this line ? Its redundant imo *Submitted by 0x421f*
- [N-02] Initialization with empty `_symbol` *Submitted by Jujic*
- [N-03] Open TODOs *Submitted by robee, also found by Meta0xNull*
- [N-04] safeApprove of openZeppelin is deprecated *Submitted by robee*
- [N-05] Solidity compiler versions mismatch *Submitted by robee*
- [N-06] `GatewayVault` events not used *Submitted by cmichel*
- [N-07] Require with empty message *Submitted by robee*
- [N-08] Missing zero-address checks on contract construction *Submitted by hyh*
- [N-09] Outdated compiler version *Submitted by WatchPug*
- [N-10] These functions can be made modifier *Submitted by cuong*qnom_

- [N-11] withdraw() Validate lpTokenAmount At Beginning of Function Can Save Gas *Submitted by MetaOxNull*

Gas Optimizations (39)

- [G-01] `+= 1` costs extra gas *Submitted by 0x0x0x*
- [G-02] Optimize `baseSupply` calculation in `_chargeFee` *Submitted by 0x0x0x*
- [G-03] Don’t cache variables used only once *Submitted by 0x0x0x*
- [G-04] Loops can be implemented more efficiently *Submitted by 0x0x0x, also found by Jujic, WatchPug, pmerkleplant, and gzeon*
- [G-05] Use immutable variables can save gas *Submitted by WatchPug, also found by robee and 0x0x0x*
- [G-06] Constant variables can be immutable (DefaultAccessControl.sol) *Submitted by yeOlde, also found by gzeon and pauliax*
- [G-07] Declaring unnecessary immutable variables as constant can save gas *Submitted by WatchPug*
- [G-08] There is no need to assign default values to variables *Submitted by 0x0x0x*
- [G-09] Gas Optimization: Use != 0 instead of > 0 *Submitted by gzeon, also found by 0x0x0x, Jujic, pmerkleplant, and yeOlde*
- [G-10] Store Interface instead of address *Submitted by 0x1f8b*
- [G-11] Don’t check contains before remove II *Submitted by 0x1f8b*
- [G-12] Skip initialization of factory address in vault governance by predicting it before hand *Submitted by 0x421f*
- [G-13] No need of separate indexing (NFT_ID => Vault Address) *Submitted by 0x421f*
- [G-14] Make deposit efficient *Submitted by 0x421f*
- [G-15] Migrate from NonFungiblePositionManager to UniV3Pool directly *Submitted by 0x421f*
- [G-16] A more efficient for loop index proceeding *Submitted by Jujic, also found by defsec and WatchPug*
- [G-17] Unnecessary checked arithmetic in for loops *Submitted by WatchPug*

- [G-18] Adding unchecked directive can save gas *Submitted by WatchPug, also found by Jujic and yeOlde*
- [G-19] ExceptionsLibrary.sol Shorten Revert Strings to Save Gas *Submitted by Meta0xNull*
- [G-20] Remove ADMIN*DELEGATE*ROLE to Save Gas *Submitted by Meta0xNull*
- [G-21] Use of _msgSender() *Submitted by defsec*
- [G-22] Use `calldata` instead of `memory` for function parameters *Submitted by defsec*
- [G-23] Use literal `2` instead of read from storage for `_vaultTokens.length` can save gas *Submitted by WatchPug*
- [G-24] Setting `uint256` variables to `0` is redundant *Submitted by WatchPug*
- [G-25] Cache external call results can save gas *Submitted by WatchPug*
- [G-26] `LpIssuer.sol#_chargeFees()` Check `if (performanceFee > 0)` can be done earlier to save gas *Submitted by WatchPug*
- [G-27] Remove unnecessary variables can make the code simpler and save some gas *Submitted by WatchPug*
- [G-28] Cache storage variables in the stack can save gas *Submitted by WatchPug*
- [G-29] Gas: `GatewayVault._pull` can skip redirected *Submitted by cmichel*
- [G-30] Gas: Cache `_pendingTokenWhitelistAdd[i]` *Submitted by cmichel*
- [G-31] Gas: Unnecessary zero writes *Submitted by cmichel*
- [G-32] Gas Optimization: Pack `Params` struct in `IProtocolGovernance` *Submitted by gzeon, also found by robee*
- [G-33] pre-calculate expressions that do not change *Submitted by pauliax*
- [G-34] Unused imports *Submitted by robee*
- [G-35] Unnecessary array boundaries check when loading an array element twice *Submitted by robee*
- [G-36] Storage double reading. Could save SLOAD *Submitted by robee*
- [G-37] Internal functions to private *Submitted by robee*
- [G-38] Public functions to external *Submitted by robee*
- [G-39] Save Gas With The Unchecked Keyword *Submitted by yeOlde*

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top