Phuture Finance contest Findings & Analysis Report

2022-6-16

Table of contents

- G-01 State variables only set in the constructor should be declared `immutable`
- G-02 State variables can be packed into fewer storage slots
- G-03 State variables should be cached in stack variables rather than re-reading them from storage
- G-04 Result of static calls should be cached in stack variables rather than re-calling storage-touching functions
- G-05 `x = x + y` is cheaper than `x += y`
- G-06 `<array>.length` should not be looked up in every loop of a `for` -loop
- G-07 `++i` / `i++` should be `unchecked{++i}` / `unchecked{++i}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops
- G-08 `require()` / `revert()` strings longer than 32 bytes cost extra gas
- G-09 Not using the named return variables when a function returns, wastes deployment gas
- G-10 Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` statement
- G-11 It costs more gas to initialize variables to zero than to let the default of zero be applied
- G-12 `++i` costs less gas than `++i`, especially when it's used in `for` -loops (`--i` / `i--` too)
- G-13 Splitting `require()` statements that use `&&` saves gas
- G-14 Usage of `uints` / `ints` smaller than 32 bytes (256 bits) incurs overhead
- G-15 Expressions for constant values such as a call to `keccak256()`, should use `immutable` rather than `constant`
- G-16 Duplicated `require()` / `revert()` checks should be refactored to a modifier or function
- G-17 `require()` or `revert()` statements that check input arguments should be at the top of the function

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 audit 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 audit contest outlined in this document, C4 conducted an analysis of the Phuture Finance smart contract system written in Solidity. The audit contest took place between April 19—April 21 2022.

Wardens

45 Wardens contributed reports to the Phuture Finance contest:

1. WatchPug (jtp and ming)
2. TrungOre
3. csanuragjain
4. IIIIIII
5. robee
6. Kenshin
7. hyh
8. cccz
9. pedroais

10. defsec
11. joestakey
12. Dravee
13. abhinavmir
14. 0xkatana
15. Tadashi
16. kenta
17. fatima_naz
18. 0xDjango
19. rayn
20. gzeon
21. 0v3rf10w
22. ellahi
23. minhquanym
24. TerrierLover
25. oyc_109
26. z3s
27. kebabsec (okkothejawa and FlameHorizon)
28. foobar
29. fatherOfBlocks
30. xpriment626
31. sseefried
32. 0xNazgul
33. Tomio
34. slywaters
35. rfa
36. windhustler
37. simon135
38. MaratCerby

39. berndartmueller
40. jah
41. peritoflores
42. reassor
43. tabish

This contest was judged by the Float Capital team: moose-code and JasoonS.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 10 unique vulnerabilities. Of these vulnerabilities, 2 received a risk rating in the category of HIGH severity and 8 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 25 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 28 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Phuture Finance contest repository, and is composed of 21 smart contracts written in the Solidity programming language and includes 1,260 lines of Solidity code.

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/non-critical.

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 (2)

[H-01] `IndexLogic` : An attacker can mint tokens for himself using assets deposited by other users

Submitted by cccz, also found by hyh, Kenshin, pedroais, and TrungOre

In the mint function of the IndexLogic contract, users are required to transfer assets to vToken in advance, and then call the mint function to mint tokens. The attacker can monitor the asset balance in the vToken contract. When the balance is greater than lastBalance, the attacker can call the mint function to mint tokens for himself.

Proof of Concept

IndexLogic.sol#L48

Recommended Mitigation Steps

Call the transferfrom function in the mint function of the IndexLogic contract to transfer the user's assets.

olivermehr (Phuture Finance) disputed

jn-lp (Phuture Finance) commented:

> We don't expect users to directly call the Mint/Burn functions on Index. Instead, they should use the Router contract, as our frontend does.

moose-code (judge) commented:

> There is no mention of the router contract in the contest documentation and this is unreasonable for wardens to know about the Router.

> `We would like wardens to focus on any core functional logic, boundary case errors or similar issues which could be utilized by an attacker to take funds away from clients who have funds deposited in the protocol.`

> This a core logic error that could be used to take funds away from clients and given there is no mention of the router and only part of the code is submitted, I am siding with the wardens on this and awarding in full.

[H-02] `UniswapV2PriceOracle.sol` `currentCumulativePrices()` will revert when `priceCumulative` addition overflow

Submitted by WatchPug

[UniswapV2PriceOracle.sol#L62](#)

```
    (uint price0Cumulative, uint price1Cumulative, uint32 blockTimes
```

Because the Solidity version used by the current implementation of `UniswapV2OracleLibrary.sol` is `>=0.8.7`, and there are some breaking changes in Solidity v0.8.0:

> Arithmetic operations revert on underflow and overflow.

Ref: [https://docs.soliditylang.org/en/v0.8.13/080-breaking-changes.html#silent-changes-of-the-semantics](#)

While in `UniswapV2OracleLibrary.sol`, subtraction overflow is desired at `blockTimestamp - blockTimestampLast` in `currentCumulativePrices()`:

https://github.com/Uniswap/v2-periphery/blob/master/contracts/libraries/UniswapV2OracleLibrary.sol#L25-L33

```
    if (blockTimestampLast != blockTimestamp) {
        // subtraction overflow is desired
        uint32 timeElapsed = blockTimestamp - blockTimestampLast;
        // addition overflow is desired
        // counterfactual
        price0Cumulative += uint(FixedPoint.fraction(reserve1, reser
        // counterfactual
        price1Cumulative += uint(FixedPoint.fraction(reserve0, reser
    }
```

In another word, `Uniswap/v2-periphery/contracts/libraries/UniswapV2OracleLibrary` only works at solidity `< 0.8.0`.

As a result, when `price0Cumulative` or `price1Cumulative` is big enough, `currentCumulativePrices` will revert due to overflow.

Impact

Since the overflow is desired in the original version, and it's broken because of using Solidity version >0.8. The `UniswapV2PriceOracle` contract will break when the desired overflow happens, and further breaks other parts of the system that relies on `UniswapV2PriceOracle`.

Recommended Mitigation Steps

Note: this recommended fix requires a fork of the library contract provided by Uniswap.

Change to:

```
    if (blockTimestampLast != blockTimestamp) {
        unchecked {
            // subtraction overflow is desired
            uint32 timeElapsed = blockTimestamp - blockTimestampLast
            // addition overflow is desired
```

```
            // counterfactual
            price0Cumulative += uint(FixedPoint.fraction(reserve1, r
            // counterfactual
            price1Cumulative += uint(FixedPoint.fraction(reserve0, r
        }
    }
```

jn-lp (Phuture Finance) confirmed and resolved

Medium Risk Findings (8)

[M-01] Index managers can rug user funds

Submitted by lllllll, also found by Kenshin

The `ORDERER_ROLE` role has the ability to arbitrarily transfer user funds, and this role is shared between both the `orderer` and people who can rebalance the index.

Even if the owner is benevolent the fact that there is a rug vector available may negatively impact the protocol's reputation. See this example where a similar finding has been flagged as a high-severity issue. I've downgraded this instance to be a medium since it requires a malicious manager.

Proof of Concept

The role is given to the `orderer` so it has the ability to add/remove funds during Uniswap operations: File: contracts/vToken.sol (lines 80-87)

```
    /// @inheritdoc IvToken
    function transferFrom(
        address _from,
        address _to,
        uint _shares
    ) external override nonReentrant onlyRole(ORDERER_ROLE) {
        _transfer(_from, _to, _shares);
    }
```

The role is also required to initiate rebalances: File: contracts/TopNMarketCapIndex.sol (lines 67-68)

```
/// @notice Reweighs index assets according to the latest ma
function reweight() external override onlyRole(ORDERER_ROLE)
```

File: contracts/TrackedIndex.sol (lines 56-57)

```
/// @notice Reweighs index assets according to the latest ma
function reweight() external override onlyRole(ORDERER_ROLE)
```

It is not necessary for the person/tool initiating reweights to also have the ability to arbitrarily transfer funds, so they should be separate roles. If the `orderer` also needs to be able to reweight, the `orderer` should also be given the new role.

Recommended Mitigation Steps

Split the role into two, and only give the `ORDERER_ROLE` role to the `orderer`.

olivermehr (Phuture Finance) disputed

jn-lp (Phuture Finance) commented:

> `ORDERER_ROLE` role is only given to Orderer contract by multisig, which must have the ability to `reweight` indices as well as to `transferFrom` on vToken contract

moose-code (judge) commented:

> I agree with the warden at the very least there is only benefit in splitting this role out appropriately into two roles. There is likely a case where the ordered and index rebalancers aren't the same.

[M-02] Chainlink's `latestRoundData` might return stale or incorrect results

*Submitted by cccz, also found by 0xDjango, Oxkatana, berndartmueller, defsec, Dravee, fatima*naz, IIIIIII, jah, kebabsec, kenta, pedroais, peritoflores, rayn, reassor, tabish, and WatchPug_

On ChainlinkPriceOracle.sol, we are using latestRoundData, but there is no check if the return value indicates stale data.

```
(, int basePrice, , , ) = baseAggregator.latestRoundData
(, int quotePrice, , , ) = assetInfo.aggregator.latestRo
```

This could lead to stale prices according to the Chainlink documentation:

https://docs.chain.link/docs/historical-price-data/#historical-rounds
https://docs.chain.link/docs/faq/#how-can-i-check-if-the-answer-to-a-round-is-being-carried-over-from-a-previous-round

Proof of Concept

ChainlinkPriceOracle.sol#L83-L84

Recommended Mitigation Steps

Consider adding missing checks for stale data.

For example:

```
(uint80 baseRoundID, int256 basePrice, , uint256 baseTimesta
(uint80 quoteRoundID, int256 quotePrice, , uint256 quoteTime
require(BaseAnsweredInRound >= baseRoundID && quoteAnsweredI
require(baseTimestamp != 0 && quoteTimestamp != 0 ,"Round no
require(basePrice > 0 && quotePrice > 0,"Chainlink answer re
```

olivermehr (Phuture Finance) confirmed

moose-code (judge) commented:

> Confirming medium issue across the board.

[M-03] Inactive skipped assets can be drained from the index

Submitted by lllllll

If an index has any inactive assets with the role `SKIPPED_ASSET_ROLE`, a user can repeatedly deposit and withdraw assets, always getting the skipped asset without having to deposit any

Proof of Concept

During minting, any asset that has the 'skipped' role is excluded from the checks of assets deposited: File: contracts/IndexLogic.sol (lines 60-70)

```
        for (uint i; i < inactiveAssets.length(); ++i) {
            if (!IAccessControl(registry).hasRole(SKIPPED_ASSET_
                uint lastBalanceInAsset = IvToken(
                    IvTokenFactory(vTokenFactory).createOrReturn
                ).lastAssetBalanceOf(address(this));
                lastAssetBalanceInBase += lastBalanceInAsset.mul
                    FixedPoint112.Q112,
                    oracle.refreshedAssetPerBaseInUQ(inactiveAss
                );
            }
        }
```

During burning, however, there's a bug that only skips if there are 'blacklisted' assets: File: contracts/IndexLogic.sol (lines 125-140)

```
        for (uint i; i < length + inactiveAssets.length(); ++i)
            address asset = i < length ? assets.at(i) : inactive
            if (containsBlacklistedAssets && IAccessControl(regi
                continue;
            }

            IvToken vToken = IvToken(IvTokenFactory(vTokenFactor
            uint indexAssetBalance = vToken.balanceOf(address(th
            uint accountBalance = (value * indexAssetBalance) /
            if (accountBalance == 0) {
                continue;
            }
```

```
// calculate index value in vault to be burned
vToken.transfer(address(vToken), accountBalance);
vToken.burn(_recipient);
```

This means that users will be passed back inactive skipped assets even if they never deposited any.

Recommended Mitigation Steps

I believe the `&&` was meant to be a `||` in the `SKIPPED_ASSET_ROLE` in the code block directly above. Changing the code to be that way would be the fix.

olivermehr (Phuture Finance) disputed

jn-lp (Phuture Finance) commented:

> That's totally expected behavior. We want to get rid of the dust of skipped assets in our index.

moose-code (judge) commented:

> Awarding the warden here since the documentation of the contest should've clearly mentioned that this is intentional behavior for skipped assets to be able to be drained. Well worth the warden bringing this up. This is well within the scope of the contest and it's possible old assets may not be dust and contain material value.

[M-04] Wrong requirement in reweight function (`ManagedIndexReweightingLogic.sol`)

Submitted by TrungOre

ManagedIndexReweightingLogic.sol#L32
IIndexRegistry.sol#L19

The list of assets won't be changed after reweight because of reverted tx.

Proof of Concept

`require(_updatedAssets.length <= IIndexRegistry(registry).maxComponents())` when reweight is not true, because as in the doc, `maxComponent` is the maximum assets for an index, but `_updatedAssets` also contain the assets that you want to remove. So the comparision makes no sense.

Recommended Mitigation Steps

Require `assets.length() <= IIndexRegistry(registry).maxComponents()` at the end of function instead.

jn-lp (Phuture Finance) confirmed and resolved

[M-05] Asset Manager can update existing `_assetAggregator`

Submitted by csanuragjain

ChainlinkPriceOracle.sol#L60

Asset Manager can update the aggregator of an existing asset thus impacting all function making use of this asset. Ideally if an aggregator is already set for an asset the function should fail.

Proof of Concept

1. Asset Manager call function addAsset to adds an asset X with assetAggregator value as Y
2. This is being utilized across application
3. Now Asset Manager calls the same function addAsset with asset X with assetAggregator value as Z
4. Asset aggregator value for asset X gets changed to Z even though it was already set to Y

Recommended Mitigation Steps

addAsset should only work if assetInfoOf[_asset] value is empty.

olivermehr (Phuture Finance) disputed

jn-lp (Phuture Finance) commented:

> Aggregators often break or are updated to new logic, the manager tracks these changes and sets the value to the current one.

moose-code (judge) commented:

> Have to assume that ASSET*MANAGER*ROLE is behaving honestly in the first place otherwise everything falls apart, so this is a centralization issue.

> The big question is who is being given the ASSET*MANAGER*ROLE ? This role has the power to rug everyone in every index.

> Given this is unclear who is given this role (can't see anything in codebase explicitly on it, no deploy scripts, no documentation on it), one can't know who is given ASSET*MANAGER*ROLE. Given this assumption, this is a valid finding as basically one asset manager could change the oracle for another asset managers index?

> Going to give this one to the warden for bringing up a valid point.

[M-06] Duplicate asset can be added

Submitted by csanuragjain

ManagedIndex.sol#L35
TopNMarketCapIndex.sol#L57
TrackedIndex.sol#L45

Initialize function can be called multiple times with same asset. Calling with same asset will make duplicate entries in assets list. Any function reading assets will get impacted and would retrieve duplicate asset

Proof of Concept

1. Observe that initialize function can be called multiple times

2. Admin calls initialize function with asset X
3. asset X gets added in assets object
4. Admin again calls initialize function with asset X
5. asset X again gets added in assets object making duplicate entries

Recommended Mitigation Steps

Add a check to fail if assets already contains the passed asset argument. Also add a modifier so that initialize could only be called once.

```
require(!assets.contain(asset), "Asset already exists");
```

olivermehr (Phuture Finance) disputed

jn-lp (Phuture Finance) commented:

> We require caller of `initialize` method to be a factory (which is non-upgradable contract), so it can't be called twice

> *see:*

```
require(msg.sender == factory, "ManagedIndex: FORBIDDEN");
```

moose-code (judge) commented:

> Given the factory contract is not supplied it makes it impossible to know these things and hence siding with the warden for the disclosure.

> " to be a factory (which is non-upgradable contract)" i.e. one can't know this if the factory is not supplied or documented.

[M-07] Tokens with fee on transfer are not supported

Submitted by robee

There are ERC20 tokens that charge fee for every transfer() / transferFrom().

Vault.sol#addValue() assumes that the received amount is the same as the transfer amount, and uses it to calculate attributions, balance amounts, etc. But, the actual transferred amount can be lower for those tokens.

Recommended Mitigation Steps

Therefore it's recommended to use the balance change before and after the transfer instead of the amount. This way you also support the tokens with transfer fee - that are popular.

IndexLogic.sol#L115

olivermehr (Phuture Finance) confirmed

[M-08] Wrong `shareChange()` function (`vToken.sol`)

Submitted by TrungOre

vToken.sol#L160

Users can get the wrong amount of vToken
=> Make users lose their fund

Proof of Concept

Base on the code in function `shareChange()` in vToken.sol
Assume that if `oldShare = totalSupply > 0`,

- `newShares`

```
= (_amountInAsset * (_totalSupply - oldShares)) / (_assetBalance - availableAssets);
= (_amountInAsset * (_totalSupply - _totalSupply)) / (_assetBalance - availableAssets);
= 0
```

It make no sense, because if `amountInAsset >> availableAssets`, `newShares` should be bigger than `oldShares`, but in this case `newShares = 0 < oldShares`

Recommended Mitigation Steps

Modify the line from `if (_totalSupply > 0)` to `if (_totalSupply - oldShares > 0)`.

olivermehr (Phuture Finance) disputed

jn-lp (Phuture Finance) commented:

> Such a case is considered impossible due to the fact that it can only work with a 0xdead address.

moose-code (judge) commented:

> Agree it's not an issue as on initialization tokens are sent to the burn address making this unlikely.

```
/// @inheritdoc IvToken
function burnFor(address _recipient) external override nonReentrant onlyRole(ORDERER_ROLE) returns (uint) {
    return _burn(_recipient);
}
```

> However the orderer role could possibly burn the tokens held by the burn address causing this issue to happen.

JasoonS (judge) commented:

> Agree with mitigation step:
>
> Modify the line from if (*totalSupply > 0) to if (*totalSupply - oldShares > 0)
>
> If it were impossible for tokens to be burned from the 0xdead address then this wouldn't be a concern.
>
> So although extremely unlikely, this is valid.

Low Risk and Non-Critical Issues

For this contest, 25 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by IIIIIII received the top score from the judge.

The following wardens also submitted reports: defsec, robee, abhinavmir, Dravee, hyh, joestakey, Tadashi, Kenshin, foobar, gzeon, 0xkatana, kenta, minhquanym, xpriment626, TerrierLover, 0v3rf10w, 0xDjango, ellahi, fatima_naz, oyc_109, rayn, sseefried, z3s, *and* kebabsec.

[L-01] `require()` should be used instead of `assert()`

1. File: contracts/IndexLogic.sol (line 72)

```
        assert(minAmountInBase != type(uint).max);
```

[L-02] Incorrect comment

Transfers the current balance if there is less available, rather than the usual reverting

1. File: contracts/vToken.sol (line 216)

```
    /// @param _amount Amount of assets to transfer
```

[L-03] Unbounded loops with external calls

The interface and the function should require a start index and a lenght, so that the index composition can be fetched in batches without running out of gas. If there are thousands of index components (e.g. like the Wilshire 5000 index), the function may revert

1. File: contracts/BaseIndex.sol (lines 75-81)

```
    function anatomy() external view override returns (address[]
        _assets = assets.values();
```

```
            _weights = new uint8[](_assets.length);
            for (uint i; i < _assets.length; ++i) {
                _weights[i] = weightOf[_assets[i]];
            }
        }
```

[L-04] Insufficient input validation

Checking for length greater than one is useless because the caller can just pass a weighting of zero for the second asset in order to exclude it

1. File: contracts/ManagedIndexReweightingLogic.sol (line 30)

```
            _updatedAssets.length > 1 &&
```

[L-05] Registries should have ability to have per-index overrides

If two indexes share the same registry, it's not possible to separately apply `SKIPPED_ASSET_ROLE` for one but not the other. It's not always clear during index creation whether there will be circumstances that affect one but not the other index

1. File: contracts/BaseIndex.sol (line 38)

```
        registry = IIndexFactory(_factory).registry();
```

[L-06] Uniswap DOS

The `README.md` talks about the fact that the orderer splits up orders to reduce price impact. This means that either the `orderer` has a slippage bounds which can DOSed with sandwich attacks, or the code uses some sort of VWAP/TWAP, which can also be gamed with flash loans submitted for every slice of the order

1. File: contracts/IndexLogic.sol (line 142)

```
IOrderer(orderer).reduceOrderAsset(asset, totalS
```

[N-01] Adding a `return` statement when the function defines a named return variable, is redundant

1. File: contracts/libraries/AUMCalculationLibrary.sol (line 71)

```
        return z_;
```

2. File: contracts/libraries/FullMath.sol (line 39)

```
            return result;
```

3. File: contracts/libraries/FullMath.sol (line 106)

```
        return result;
```

[N-02] `require()` / `revert()` statements should have descriptive reason strings

1. File: contracts/libraries/FullMath.sol (line 35)

```
            require(denominator > 0);
```

2. File: contracts/libraries/FullMath.sol (line 44)

```
        require(denominator > prod1);
```

3. File: contracts/libraries/FullMath.sol (line 123)

```
require(result < type(uint256).max);
```

[N-03] `constant` s should be defined rather than using magic numbers

1. File: contracts/IndexLogic.sol (line 82)

```
_mint(address(0xdead), IndexLibrary.INITIAL_QUANTITY
```

2. File: contracts/libraries/FullMath.sol (line 88)

```
uint256 inv = (3 * denominator) ^ 2;
```

[N-04] Use bit shifts in an imutable variable rather than long bit masks of a single bit, for readability

1. File: contracts/libraries/FixedPoint112.sol (line 10)

```
uint256 internal constant Q112 = 0x10000000000000000000000000
```

[N-05] Use a more recent version of solidity

Use a solidity version of at least 0.8.12 to get `string.concat()` to be used instead of `abi.encodePacked(<str>,<str>)`

1. File: contracts/ManagedIndex.sol (line 3)

```
pragma solidity >=0.8.7;
```

[N-06] Variable names that consist of all capital letters should be reserved for `const` / `immutable` variables

If the variable needs to be different based on which class it comes from, a `view` / `pure` *function* should be used instead (e.g. like this).

1. File: contracts/ManagedIndex.sol (line 17)

```
bytes32 private REWEIGHT_INDEX_ROLE;
```

2. File: contracts/vToken.sol (line 41)

```
NAV.Data internal _NAV;
```

[N-07] File is missing NatSpec

1. File: contracts/interfaces/external/IChainLinkFeed.sol (line 0)

```
// SPDX-License-Identifier: GPL-2.0-or-later
```

2. File: contracts/interfaces/external/IWETH.sol (line 0)

```
// SPDX-License-Identifier: GPL-2.0-or-later
```

[N-08] NatSpec is incomplete

1. File: contracts/interfaces/IChainlinkPriceOracle.sol (lines 10-13)

```
/// @notice Adds `_asset` to the oracle
/// @param _asset Asset's address
/// @param _asset Asset aggregator's address
function addAsset(address _asset, address _assetAggregator)
```

Missing: `@param _assetAggregator`

2. File: contracts/interfaces/IFeePool.sol (lines 8-10)

```
/// @notice Minting fee in base point format
/// @return Returns minting fee in base point (BP) format
function mintingFeeInBPOf(address _index) external view retu
```

Missing: `@param _index`

3. File: contracts/interfaces/IFeePool.sol (lines 12-14)

```
/// @notice Burning fee in base point format
/// @return Returns burning fee in base point (BP) format
function burningFeeInBPOf(address _index) external view retu
```

Missing: `@param _index`

4. File: contracts/interfaces/IFeePool.sol (lines 16-18)

```
/// @notice AUM scaled per seconds rate
/// @return Returns AUM scaled per seconds rate
function AUMScaledPerSecondsRateOf(address _index) external
```

Missing: `@param _index`

5. File: contracts/interfaces/IPriceOracle.sol (lines 8-10)

```
/// @notice Updates and returns asset per base
/// @return Asset per base in UQ
function refreshedAssetPerBaseInUQ(address _asset) external
```

Missing: `@param _asset`

6. File: contracts/interfaces/IPriceOracle.sol (lines 12-14)

```
/// @notice Returns last asset per base
/// @return Asset per base in UQ
```

```
function lastAssetPerBaseInUQ(address _asset) external view
```

Missing: `@param _asset`

7. File: contracts/interfaces/IvTokenFactory.sol (lines 8-10)

```
/// @notice Creates or returns address of previously createc
/// @param _asset Asset to create or return vToken for
function createOrReturnVTokenOf(address _asset) external ret
```

Missing: `@return`

8. File: contracts/interfaces/IvToken.sol (lines 72-74)

```
/// @notice Returns amount of assets for the given account w
/// @return Amount of assets for the given account with the
function assetDataOf(address _account, uint _shares) externa
```

Missing: `@param _account` `@param _shares`

9. File: contracts/libraries/NAV.sol (lines 18-26)

```
/// @notice Transfer `_amount` of shares between given addre
/// @param _from Account to send shares from
/// @param _to Account to send shares to
/// @param _amount Amount of shares to send
function transfer(
    Data storage self,
    address _from,
    address _to,
    uint _amount
```

Missing: `@param self`

10. File: contracts/libraries/NAV.sol (lines 34-40)

```
        /// @param _balance New shares maximum limit
        /// @param _recipient Recipient that will receive minted sha
        function mint(
            Data storage self,
            uint _balance,
            address _recipient
        ) internal returns (uint shares) {
```

Missing: `@return`

11. File: contracts/libraries/NAV.sol (lines 54-56)

```
        /// @param self Data structure reference
        /// @param _balance Shares balance
        function burn(Data storage self, uint _balance) internal ret
```

Missing: `@return`

[N-09] Event is missing `indexed` fields

Each `event` should use three `indexed` fields if there are three or more fields

1. File: contracts/interfaces/IAnatomyUpdater.sol (line 8)

```
        event UpdateAnatomy(address asset, uint8 weight);
```

2. File: contracts/interfaces/IvToken.sol (line 13)

```
        event VTokenTransfer(address indexed from, address indexed t
```

[N-10] Typos

1. File: contracts/interfaces/IAnatomyUpdater.sol (line 6)

```
    /// @notice Contains event for aatomy update
```

aatomy

2. File: contracts/interfaces/IReweightableIndex.sol (line 5)

```
    /// @title Rewightable index interface
```

Rewightable

3. File: contracts/libraries/FullMath.sol (line 101)

```
                // correct result modulo 2**256. Since the precoditi
```

preconditions

4. File: contracts/vToken.sol (line 84)

Why is this one named `_shares` whereas the others are named `_amount`

```
            uint _shares
```

[N-11] Use of sensitive/non-inclusive terms

Rename to `containsBlockedAssets`

1. File: contracts/IndexLogic.sol (line 101)

```
            bool containsBlacklistedAssets;
```

moose-code (judge) commented:

> Excellent report. Well thought out, deep understanding.

Gas Optimizations

For this contest, 28 reports were submitted by wardens detailing gas optimizations. The report highlighted below by IIIIIII received the top score from the judge.

The following wardens also submitted reports: defsec, joestakey, Dravee, fatherOfBlocks, robee, 0xNazgul, 0v3rf10w, 0xkatana, Kenshin, Tomio, ellahi, TrungOre, fatima_naz, gzeon, kenta, oyc_109, slywaters, rfa, windhustler, Tadashi, TerrierLover, minhquanym, simon135, z3s, 0xDjango, MaratCerby, *and* rayn.

[G-01] State variables only set in the constructor should be declared `immutable`

Avoids a Gsset (20000 gas)

1. File: contracts/ManagedIndex.sol (line 17)

```
bytes32 private REWEIGHT_INDEX_ROLE;
```

2. File: contracts/PhuturePriceOracle.sol (line 24)

```
address public base;
```

3. File: contracts/PhuturePriceOracle.sol (line 27)

```
address public registry;
```

4. File: contracts/PhuturePriceOracle.sol (line 33)

```
uint8 private baseDecimals;
```

[G-02] State variables can be packed into fewer storage slots

If variables occupying the same slot are both written the same function or by the constructor, avoids a separate Gsset (20000 gas). Reads of the variables are also cheaper

1. File: contracts/PhuturePriceOracle.sol (line 24)

```
        address public base;
```

Variable ordering with 3 slots instead of the current 4: mapping(32):priceOracleOf, address(20):base, uint8(1):baseDecimals, address(20):registry

[G-03] State variables should be cached in stack variables rather than re-reading them from storage

The instances below point to the second access of a state variable within a function. Caching will replace each Gwarmaccess (100 gas) with a much cheaper stack read. Less obvious optimizations include having local storage variables of mappings within state variable mappings or mappings within state variable structs, having local storage variables of structs within mappings, or having local caches of state variable contracts/addresses.

1. File: contracts/PhuturePriceOracle.sol (line 84)

```
            return IPriceOracle(priceOracleOf[_asset]).refreshedAsse
```

2. File: contracts/PhuturePriceOracle.sol (line 94)

```
            return IPriceOracle(priceOracleOf[_asset]).lastAssetPerI
```

3. File: contracts/UniswapV2PathPriceOracle.sol (line 35)

(save `asset` pointer for next iteration of the loop)

```
address asset = path[i + 1];
```

4. File: contracts/UniswapV2PathPriceOracle.sol (line 50)

(save `asset` pointer for next iteration of the loop)

```
address asset = path[i + 1];
```

5. File: contracts/UniswapV2PriceOracle.sol (line 51)

```
uint32 timeElapsed = blockTimestamp - blockTimestampLast
```

6. File: contracts/vToken.sol (line 219)

```
IERC20(asset).safeTransfer(_recipient, Math.min(_amount,
```

[G-04] Result of static calls should be cached in stack variables rather than re-calling storage-touching functions

Caching will replace each Gwarmaccess (100 gas) with a much cheaper stack read.

1. File: contracts/IndexLogic.sol (line 41)

```
if (weightOf[assets.at(i)] == 0) {
```

2. File: contracts/ManagedIndexReweightingLogic.sol (line 40)

```
uint availableAssets = IvToken(IvTokenFactory(vToker
```

3. File: contracts/TopNMarketCapReweightingLogic.sol (line 39)

```
uint availableAssets = IvToken(IvTokenFactory(vToker
```

[G-05] `x = x + y` is cheaper than `x += y`

1. File: contracts/libraries/NAV.sol (line 28)

```
self.balanceOf[_from] -= _amount;
```

2. File: contracts/libraries/NAV.sol (line 29)

```
self.balanceOf[_to] += _amount;
```

[G-06] `<array>.length` should not be looked up in every loop of a `for`-loop

Even memory arrays incur the overhead of bit tests and bit shifts to calculate the array length. Storage array length checks incur an extra Gwarmaccess (100 gas) PER-LOOP.

1. File: contracts/BaseIndex.sol (line 78)

```
for (uint i; i < _assets.length; ++i) {
```

2. File: contracts/ManagedIndexReweightingLogic.sol (line 50)

```
for (uint i; i < _updatedAssets.length; ++i) {
```

3. File: contracts/ManagedIndexReweightingLogic.sol (line 96)

```
for (uint i; i < _inactiveAssets.length; ++i) {
```

4. File: contracts/ManagedIndex.sol (line 30)

```
        for (uint i; i < _assets.length; ++i) {
```

5. File: contracts/TopNMarketCapIndex.sol (line 48)

```
        for (uint i; i < _assets.length; ++i) {
```

6. File: contracts/TopNMarketCapReweightingLogic.sol (line 104)

```
        for (uint i; i < _inactiveAssets.length; ++i) {
```

7. File: contracts/TrackedIndex.sol (line 35)

```
        for (uint i; i < _assets.length; ++i) {
```

[G-07] `++i` / `i++` should be `unchecked{++i}` / `unchecked{++i}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops

1. File: contracts/BaseIndex.sol (line 78)

```
        for (uint i; i < _assets.length; ++i) {
```

2. File: contracts/IndexLogic.sol (line 39)

```
        for (uint i; i < assets.length(); ++i) {
```

3. File: contracts/IndexLogic.sol (line 60)

```
for (uint i; i < inactiveAssets.length(); ++i) {
```

4. File: contracts/IndexLogic.sol (line 102)

```
for (uint i; i < length; ++i) {
```

5. File: contracts/IndexLogic.sol (line 125)

```
for (uint i; i < length + inactiveAssets.length(); ++i)
```

6. File: contracts/ManagedIndexReweightingLogic.sol (line 38)

```
for (uint i; i < assets.length(); ++i) {
```

7. File: contracts/ManagedIndexReweightingLogic.sol (line 50)

```
for (uint i; i < _updatedAssets.length; ++i) {
```

8. File: contracts/ManagedIndexReweightingLogic.sol (line 96)

```
for (uint i; i < _inactiveAssets.length; ++i) {
```

9. File: contracts/ManagedIndex.sol (line 30)

```
for (uint i; i < _assets.length; ++i) {
```

10. File: contracts/TopNMarketCapIndex.sol (line 48)

```
for (uint i; i < _assets.length; ++i) {
```

11. File: contracts/TopNMarketCapReweightingLogic.sol (line 37)

```
        for (uint i; i < assets.length(); ++i) {
```

12. File: contracts/TopNMarketCapReweightingLogic.sol (line 51)

```
        for (uint _i; _i < diff.assetCount; ++_i) {
```

13. File: contracts/TopNMarketCapReweightingLogic.sol (line 104)

```
        for (uint i; i < _inactiveAssets.length; ++i) {
```

14. File: contracts/TrackedIndexReweightingLogic.sol (line 37)

```
        for (uint i; i < assets.length(); ++i) {
```

15. File: contracts/TrackedIndexReweightingLogic.sol (line 66)

```
        for (uint i; i < assets.length(); ++i) {
```

16. File: contracts/TrackedIndex.sol (line 35)

```
        for (uint i; i < _assets.length; ++i) {
```

17. File: contracts/UniswapV2PathPriceOracle.sol (line 34)

```
        for (uint i = 0; i < path.length - 1; i++) {
```

18. File: contracts/UniswapV2PathPriceOracle.sol (line 49)

```
for (uint i = 0; i < path.length - 1; i++) {
```

[G-08] `require()` / `revert()` strings longer than 32 bytes cost extra gas

1. File: contracts/TopNMarketCapIndex.sol (line 74)

```
revert("TopNMarketCapIndex: REWEIGH_FAILED");
```

2. File: contracts/TopNMarketCapReweightingLogic.sol (line 67)

```
require(IAccessControl(registry).hasRole(ASSET_F
```

3. File: contracts/UniswapV2PathPriceOracle.sol (line 25)

```
require(_oracles.length == _path.length - 1, "UniswapV2F
```

[G-09] Not using the named return variables when a function returns, wastes deployment gas

1. File: contracts/vToken.sol (line 91)

```
return _mint(msg.sender);
```

2. File: contracts/vToken.sol (line 96)

```
return _burn(_recipient);
```

[G-10] Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` statement

1. File: contracts/IndexLogic.sol (line 76)

```
            require(lastAssetBalanceInBase > 0, "Index: INSUFFIC
```

2. File: contracts/IndexLogic.sol (line 98)

```
        require(value > 0, "Index: INSUFFICIENT_AMOUNT");
```

3. File: contracts/libraries/FullMath.sol (line 35)

```
            require(denominator > 0);
```

4. File: contracts/libraries/IndexLibrary.sol (line 29)

```
        require(_assetPerBaseInUQ > 0, "IndexLibrary: ORACLE");
```

5. File: contracts/libraries/NAV.sol (line 49)

```
        require(shares > 0, "NAV: INSUFFICIENT_AMOUNT");
```

6. File: contracts/libraries/NAV.sol (line 59)

```
        require(amount > 0, "NAV: INSUFFICIENT_SHARES_BURNED");
```

[G-11] It costs more gas to initialize variables to zero than to let the default of zero be applied

1. File: contracts/UniswapV2PathPriceOracle.sol (line 34)

```
        for (uint i = 0; i < path.length - 1; i++) {
```

2. File: contracts/UniswapV2PathPriceOracle.sol (line 49)

```
        for (uint i = 0; i < path.length - 1; i++) {
```

[G-12] `++i` costs less gas than `++i`, especially when it's used in `for`-loops (`--i` / `i--` too)

1. File: contracts/UniswapV2PathPriceOracle.sol (line 34)

```
        for (uint i = 0; i < path.length - 1; i++) {
```

2. File: contracts/UniswapV2PathPriceOracle.sol (line 49)

```
        for (uint i = 0; i < path.length - 1; i++) {
```

[G-13] Splitting `require()` statements that use `&&` saves gas

See this issue for an example

1. File: contracts/ChainlinkPriceOracle.sol (line 51)

```
        require(_baseAggregator != address(0) && _base != addres
```

2. File: contracts/ChainlinkPriceOracle.sol (line 86)

```
        require(basePrice > 0 && quotePrice > 0, "ChainlinkPrice
```

3. File: contracts/ManagedIndexReweightingLogic.sol (lines 29-34)

```
        require(
            _updatedAssets.length > 1 &&
                _updatedWeights.length == _updatedAssets.length
```

```
                _updatedAssets.length <= IIndexRegistry(registry
            "ManagedIndex: INVALID"
        );
```

4. File: contracts/UniswapV2PriceOracle.sol (line 46)

```
        require(reserve0 != 0 && reserve1 != 0, "UniswapV2PriceO
```

[G-14] Usage of `uints` / `ints` smaller than 32 bytes (256 bits) incurs overhead

> When using elements that are smaller than 32 bytes, your contract's gas usage may be higher. This is because the EVM operates on 32 bytes at a time. Therefore, if the element is smaller than that, the EVM must use more operations in order to reduce the size of the element from 32 bytes to the desired size.

https://docs.soliditylang.org/en/v0.8.11/internals/layout_in_storage.html
Use a larger size then downcast where needed

See original submission for instances.

[G-15] Expressions for constant values such as a call to `keccak256()`, should use `immutable` rather than `constant`

See this issue for a detail description of the issue

1. File: contracts/BaseIndex.sol (line 25)

```
    bytes32 internal constant INDEX_MANAGER_ROLE = keccak256("IN
```

2. File: contracts/ChainlinkPriceOracle.sol (line 29)

```
    bytes32 private constant ASSET_MANAGER_ROLE = keccak256("ASS
```

3. File: contracts/IndexLogic.sol (line 25)

```
bytes32 internal constant ASSET_ROLE = keccak256("ASSET_ROLE
```

4. File: contracts/IndexLogic.sol (line 27)

```
bytes32 internal constant SKIPPED_ASSET_ROLE = keccak256("SK
```

5. File: contracts/ManagedIndexReweightingLogic.sol (line 25)

```
bytes32 internal constant ASSET_ROLE = keccak256("ASSET_ROLE
```

6. File: contracts/PhuturePriceOracle.sol (line 21)

```
bytes32 private constant ASSET_MANAGER_ROLE = keccak256("ASS
```

7. File: contracts/TopNMarketCapIndex.sol (line 18)

```
bytes32 internal constant ORDERER_ROLE = keccak256("ORDERER_
```

8. File: contracts/TopNMarketCapReweightingLogic.sol (line 27)

```
bytes32 internal constant ASSET_ROLE = keccak256("ASSET_ROLE
```

9. File: contracts/TrackedIndexReweightingLogic.sol (line 25)

```
bytes32 internal constant ASSET_ROLE = keccak256("ASSET_ROLE
```

10. File: contracts/TrackedIndex.sol (line 17)

```
bytes32 internal constant ORDERER_ROLE = keccak256("ORDERER_
```

11. File: contracts/vToken.sol (line 27)

```
bytes32 private constant INDEX_ROLE = keccak256("INDEX_ROLE'
```

12. File: contracts/vToken.sol (line 29)

```
bytes32 private constant ORACLE_ROLE = keccak256("ORACLE_ROI
```

13. File: contracts/vToken.sol (line 31)

```
bytes32 private constant ORDERER_ROLE = keccak256("ORDERER_F
```

14. File: contracts/vToken.sol (line 33)

```
bytes32 private constant RESERVE_MANAGER_ROLE = keccak256("F
```

[G-16] Duplicated `require()` / `revert()` checks should be refactored to a modifier or function

1. File: contracts/PhuturePriceOracle.sol (line 83)

```
        require(priceOracleOf[_asset] != address(0), "PhuturePri
```

[G-17] `require()` or `revert()` statements that check input arguments should be at the top of the function

Checks that involve constants should come before checks that involve state variables

1. File: contracts/ChainlinkPriceOracle.sol (line 62)

```
require(_asset != address(0), "ChainlinkPriceOracle: ZEI
```

2. File: contracts/PhuturePriceOracle.sol (line 47)

```
require(_base != address(0), "PhuturePriceOracle: ZERO")
```

3. File: contracts/vToken.sol (line 60)

```
require(_asset != address(0), "vToken: ZERO");
```

[G-18] Use custom errors rather than `revert()` / `require()` strings to save deployment gas

1. File: contracts/BaseIndex.sol (Various lines throughout the file)
2. File: contracts/ChainlinkPriceOracle.sol (Various lines throughout the file)
3. File: contracts/IndexLogic.sol (Various lines throughout the file)
4. File: contracts/libraries/FullMath.sol (Various lines throughout the file)
5. File: contracts/libraries/IndexLibrary.sol (Various lines throughout the file)
6. File: contracts/libraries/NAV.sol (Various lines throughout the file)
7. File: contracts/ManagedIndexReweightingLogic.sol (Various lines throughout the file)
8. File: contracts/ManagedIndex.sol (Various lines throughout the file)
9. File: contracts/PhuturePriceOracle.sol (Various lines throughout the file)
10. File: contracts/TopNMarketCapIndex.sol (Various lines throughout the file)
11. File: contracts/TopNMarketCapReweightingLogic.sol (Various lines throughout the file)
12. File: contracts/TrackedIndexReweightingLogic.sol (Various lines throughout the file)
13. File: contracts/TrackedIndex.sol (Various lines throughout the file)

14. File: contracts/UniswapV2PathPriceOracle.sol (Various lines throughout the file)
15. File: contracts/UniswapV2PriceOracle.sol (Various lines throughout the file)
16. File: contracts/vToken.sol (Various lines throughout the file)

[G-19] Functions guaranteed to revert when called by normal users can be marked `payable`

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as `payable` will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided.

1. File: contracts/PhuturePriceOracle.sol (line 55)

```
function setOracleOf(address _asset, address _oracle) exterr
```

2. File: contracts/PhuturePriceOracle.sol (line 62)

```
function removeOracleOf(address _asset) external override or
```

3. File: contracts/TopNMarketCapIndex.sol (line 68)

```
function reweight() external override onlyRole(ORDERER_ROLE)
```

4. File: contracts/TrackedIndex.sol (line 57)

```
function reweight() external override onlyRole(ORDERER_ROLE)
```

5. File: contracts/vToken.sol (lines 81-85)

```
function transferFrom(
    address _from,
    address _to,
```

```
        uint _shares
    ) external override nonReentrant onlyRole(ORDERER_ROLE) {
```

6. File: contracts/vToken.sol (line 90)

```
    function mint() external override nonReentrant onlyRole(INDE
```

7. File: contracts/vToken.sol (line 95)

```
    function burn(address _recipient) external override nonReent
```

8. File: contracts/vToken.sol (line 100)

```
    function mintFor(address _recipient) external override nonRe
```

9. File: contracts/vToken.sol (line 105)

```
    function burnFor(address _recipient) external override nonRe
```

[G-20] Use a more recent version of solidity

Use a solidity version of at least 0.8.10 to have external calls skip contract existence checks if the external call has a return value

See **original submission** for instances.

jn-lp (Phuture Finance) commented:

> More than half of the issues were very helpful, thanks!

moose-code (judge) commented:

> Very comprehensive, lots of good things here!

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