Beanstalk BIP-38 Audit Report

Prepared by Cyfrin

Version 2.3

Lead Auditors

Giovanni Di Siena

Carlos Amarante

October 13, 2023

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

Beanstalk is a permissionless algorithmic stablecoin protocol built on Ethereum. The protocol uses a novel dynamic peg maintenance mechanism to have the price of 1 BEAN (the Beanstalk stablecoin) continuously cross its peg value of 1 USD without centralization or collateral requirements.

The BIP-38 upgrade specifically involves migration of the BEAN:3CRV LP Tokens that underly the Unripe BEAN:3CRV token to BEAN:ETH Well LP Tokens.

The main changes introduced in the diff between the two commits referenced in the audit scope include:

- Replaced Curve integration with Wells integration.
- Removed the ability of Beanstalk DAO to arbitrarily add fertilizer by removing `FertilizerFacet::addFertilizerOwner`.
- Added `UnripeFacet::addMigratedUnderlying` to allow for liquidity migration.
- Added a requirement to `UnripeFacet::chop` that there should be an `underlyingToken` balance to allow chopping, in order to protect users from accidentally exchanging unripe LP for nothing during migration.
- Updated constant values to match on-chain addresses: `C.UNIV3_ETH_USDC_POOL`; `C.UNIV3_ETH_USDT_POOL`; `C.BEANSTALK_PUMP`; `C.BEAN_ETH_WELL`; `DepotFacet::PIPELINE`.
- Changed how Chainlink USDC and USDT Oracles are integrated into the Beanstalk protocol through `LibEthUsdOracle::getEthUsdPrice`.
- Fertilizer is now minted using WETH instead of USDC.
- Added `InitBipBasinIntegration`:
  - Updates earned Stalk per BDV per Season for BEAN and urBEAN3CRV.
  - Whitelists BEAN:ETH Well LP.
- Added `InitMigrateUnripeBean3CrvToBeanEth` which initiates the migration process.

- Modified `MetadataFacet` and `MetadataImage`.
- Pipeline is now able to receive ETH.
- Depot is now able to receive ETH.
- `LibStrings` now supports the string representation of `int256` values less than zero.
- Removed legacy support for withdrawals in `Weather::handleRain`.
- `BeanstalkPrice::price` now considers the price provided by the BEAN:ETH Well and Curve to provide a weighted average price.
- Replaced Curve integration with Well in multiple protocol components, including:
  - `LibFertilizer`
  - `LibUnripeConvert`
  - `BDVFacet`

5 Audit Scope

Cyfrin conducted an audit of Beanstalk based on the code present in the repository commit hash 12c608a, specifically the diff between this commit hash and c7a20e5 which largely pertains to the BIP-38 migration of the token underlying unripe LP from BEAN:3CRV MetaPool LP to unripe BEAN:ETH Well LP. Other changes introduced in the diff between these two commits, since the previous audit performed by Cyfrin, are also considered in the scope of this audit.

Mitigation remedations have been provided by the Beanstalk Farms team in the commit hash 7606673 contained within the pull request linked above.

6 Executive Summary

Over the course of 11 days, the Cyfrin team conducted an audit on the Beanstalk BIP-38 smart contracts provided by Beanstalk Farms. In this period, a total of 16 issues were found.

The BIP-38 migration of the token underlying unripe LP from BEAN:3CRV LP to BEAN:ETH Well LP is well written and architected. We have verified that suspension of relevant functions will work as intended, preventing users of the Protocol from losing assets when performing these actions (chop/deposit/enroot/convert) at the time of upgrade. This has also been tested in the "Interactions with Unripe fail" block of ‘Bean3CrvToBeanEThMigration.test.js‘.

Most changes are clearly documented, although there are certain areas such as the modification to `LibEthUsdOracle::getEthUsdPrice` that could benefit from additional inline comments. It is also not currently finalized how the swap from 3CRV to WETH will be performed, whether through a DEX aggregator with MEV protection or via an OTC swap. It is understood that the BCM will ensure the proper use of slippage parameters when removing/adding liquidity, and it is essential that this is the case.

Two findings related to the recaptialization of unripe LP have been raised, introduced as a result of these changes. They should be addressed to mitigate potential denial-of-service attacks on the Beanstalk Sunrise mechanism and errors in recapitalization accounting.

Summary

Project Name	Beanstalk BIP-38
Repository	Beanstalk
Commit	12c608a22535...
Audit Timeline	Sep 12th - Sep 26th
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	1
Medium Risk	1
Low Risk	1
Informational	13
Gas Optimizations	0
Total Issues	16

Summary of Findings

[H-1] Migration of unripe LP from BEAN:3CRV to BEAN:ETH does not account for recapitalization accounting error	Acknowledged
[M-1] Insufficient validation of new Fertilizer IDs allow for a denial-of-service (DoS) attack on `SeasonFacet::gm` when above peg, once the last element in the FIFO is paid	Resolved
[L-1] Incorrect handling of commas and quotation marks in the attributes of `MetadataFacet::uri`	Resolved
[I-01] Resetting of `withdrawSeasons` state was not executed on-chain as part of the BIP-36 upgrade	Resolved
[I-02] Changes to initialization contracts are not recommended after they are executed on-chain	Acknowledged
[I-03] Lack of slippage protection when removing liquidity from BEAN:3CRV MetaPool and adding liquidity to BEAN:ETH Well could result in loss of funds due to sandwich attack	Acknowledged
[I-04] `LibEthUsdOracle::getEthUsdPrice` design changes should be documented	Resolved
[I-05] Logic in `LibFertilizer::push` related to (deprecated) intermediate addition of Fertilizer to FIFO list should be removed	Acknowledged
[I-06] Consider moving the `MetadataFacet::uri` disclaimer from metadata attributes to the description	Acknowledged
[I-07] Incorrect comment in `MetadataImage::sciNotation` should be corrected	Resolved

[I-08] Continued reference to "Seeds" in `InitBipBasinIntegration::init` is confusing	Resolved
[I-09] `InitBipBasinIntegration` NatSpec title tag is inconsistent with the file/contract name	Resolved
[I-10] Conditional block in `WellPrice::getConstantProductWell` can be removed	Resolved
[I-11] Unsafe cast in `WellPrice::getDeltaB`	Resolved
[I-12] Typo in `FertilizerFacet::getMintFertilizerOut` NatSpec	Resolved
[I-13] Typo in comment within `LibSilo::_mow`	Resolved

7 Findings

7.1 High Risk

7.1.1 Migration of unripe LP from BEAN:3CRV to BEAN:ETH does not account for recapitalization accounting error

Description: The global `AppStorage::recapitalized` state refers to the dollar amount recapitalized when Fertilizer was bought with USDC and paired with BEAN for BEAN:3CRV LP. When removing this underlying liquidity and swapping 3CRV for WETH during the migration of unripe LP, it is very likely that the BCM will experience some slippage. This is more likely to be the case if the swap is made on the open market rather than an OTC deal, but either way it is likely that the dollar value of the resulting WETH, and hence BEAN:ETH LP, will be less than it was as BEAN:3CRV before the migration. Currently, `UnripeFacet::addMigratedUnderlying` updates the BEAN:ETH LP token balance underlying the unripe LP, completing the migration, but does not account for any changes in the dollar value as outlined above. Based on the current implementation, it is very likely that the BCM will complete migration by transferring less in dollar value while the recapitalization status remains the same, causing inconsistency in `LibUnripe::percentLPRecapped` and `LibUnripe::add/removeUnderlying` which are used in the conversion of urBEAN urBEANETH in `LibUnripeConvert`. Therefore, the global recapitalized state should be updated to reflect the true dollar value of recapitalization on completion of the migration.

Impact: Once sufficiently funded by purchasers of Fertilizer, it is possible that recapitalization could be considered completed with insufficient underlying BEAN:ETH LP. This amounts to a loss of user funds since the true recapitalized amount will be less than that specified by `C::dollarPerUnripeLP` which is used to calculate the total dollar liability in `LibFertilizer::remainingRecapitalization`.

Recommended Mitigation: Reassign `s.recapitalized` to the oracle USD amount of the new BEAN:ETH LP at the time of migration completion.

```
    function addMigratedUnderlying(address unripeToken, uint256 amount) external payable nonReentrant {
        LibDiamond.enforceIsContractOwner();
        IERC20(s.u[unripeToken].underlyingToken).safeTransferFrom(
            msg.sender,
            address(this),
            amount
        );
        LibUnripe.incrementUnderlying(unripeToken, amount);

+       uint256 recapitalized = amount.mul(LibEthUsdOracle.getEthUsdPrice()).div(1e18);
+       require(recapitalized != 0, "UnripeFacet: cannot calculate recapitalized");
+       s.recapitalized = s.recapitalized.add(recapitalized);
    }
```

Beanstalk Farms: This is intentional – the cost of slippage goes to the Unripe LP token holders. This should be clearly stated in the BIP draft.

Cyfrin: Acknowledged.

7.2 Medium Risk

7.2.1 Insufficient validation of new Fertilizer IDs allow for a denial-of-service (DoS) attack on `SeasonFacet::gm` when above peg, once the last element in the FIFO is paid

Description: A Fertilizer NFT can be interpreted as a bond without an expiration date which is to be repaid in Beans and includes interest (Humidity). This bond is placed in a FIFO list and intended to recapitalize the $77 million in liquidity stolen during the April 2022 exploit. One Fertilizer can be purchased for 1 USD worth of WETH: prior to BIP-38, this purchase was made using USDC.

Each fertilizer is identified by an Id that depends on `s.bpf`, indicating the cumulative amount of Beans paid per Fertilizer. This value increases each time `Sun::rewardToFertilizer` is called, invoked by `SeasonFacet::gm` if the Bean price is above peg. Therefore, Fertilizer IDs depend on `s.bpf` at the moment of minting, in addition to the amount of Beans to be paid.

The FIFO list has following components:

- `s.fFirst`: Fertilizer Id corresponding to the next Fertilizer to be paid.
- `s.fLast`: The highest active Fertilizer Id which is the last Fertilizer to be paid.
- `s.nextFid`: Mapping from Fertilizer Id to Fertilizer id, indicating the next element of a linked list. If an Id points to 0, then there is no next element.

Methods related to this FIFO list include: `LibFertilizer::push`: Add an element to the FIFO list. `LibFertilizer::setNext`: Given a fertilizer id, add a pointer to next element in the list `LibFertilizer::getNext`: Get next element in the list.

The intended behaviour of this list is to add a new element to its end whenever a new fertilizer is minted with a new Id. Intermediate addition to the list was formerly allowed only by the Beanstalk DAO, but this functionality has since been deprecated in the current upgrade with the removal of `FertilizerFacet::addFertilizerOwner`.

Consequences of replacing BEAN:3CRV MetaPool with the BEAN:ETH Well: Before this upgrade, addition of 0 Fertilizer through `LibFertilizer::addFertilizer` was impossible due to the dependency on Curve in `LibFertilizer::addUnderlying`:

```
// Previous code

    function addUnderlying(uint256 amount, uint256 minAmountOut) internal {
        //...
        C.bean().mint(
            address(this),
            newDepositedBeans.add(newDepositedLPBeans)
        );

        // Add Liquidity
        uint256 newLP = C.curveZap().add_liquidity(
            C.CURVE_BEAN_METAPOOL, // where to add liquidity
            [
                newDepositedLPBeans, // BEANS to add
                0,
                amount, // USDC to add
                0
            ], // how much of each token to add
            minAmountOut // min lp ampount to receive
        ); // @audit-ok Does not admit depositing 0 -->
↪   https://etherscan.io/address/0x5F890841f657d90E081bAbdB532A05996Af79Fe6#code#L487

        // Increment underlying balances of Unripe Tokens
        LibUnripe.incrementUnderlying(C.UNRIPE_BEAN, newDepositedBeans);
        LibUnripe.incrementUnderlying(C.UNRIPE_LP, newLP);

        s.recapitalized = s.recapitalized.add(amount);
    }
```

However, with the change of dependency involved in the Wells integration, this restriction no longer holds:

```
    function addUnderlying(uint256 usdAmount, uint256 minAmountOut) internal {
        AppStorage storage s = LibAppStorage.diamondStorage();
        // Calculate how many new Deposited Beans will be minted
        uint256 percentToFill = usdAmount.mul(C.precision()).div(
            remainingRecapitalization()
        );
        uint256 newDepositedBeans;
        if (C.unripeBean().totalSupply() > s.u[C.UNRIPE_BEAN].balanceOfUnderlying) {
            newDepositedBeans = (C.unripeBean().totalSupply()).sub(
                s.u[C.UNRIPE_BEAN].balanceOfUnderlying
            );
            newDepositedBeans = newDepositedBeans.mul(percentToFill).div(
                C.precision()
            );
        }

        // Calculate how many Beans to add as LP
        uint256 newDepositedLPBeans = usdAmount.mul(C.exploitAddLPRatio()).div(
            DECIMALS
        );

        // Mint the Deposited Beans to Beanstalk.
        C.bean().mint(
            address(this),
            newDepositedBeans
        );

        // Mint the LP Beans to the Well to sync.
        C.bean().mint(
            address(C.BEAN_ETH_WELL),
            newDepositedLPBeans
        );

        // @audit If nothing was previously deposited this function returns 0, IT DOES NOT REVERT
        uint256 newLP = IWell(C.BEAN_ETH_WELL).sync(
            address(this),
            minAmountOut
        );

        // Increment underlying balances of Unripe Tokens
        LibUnripe.incrementUnderlying(C.UNRIPE_BEAN, newDepositedBeans);
        LibUnripe.incrementUnderlying(C.UNRIPE_LP, newLP);

        s.recapitalized = s.recapitalized.add(usdAmount);
    }
```

Given that the new integration does not revert when attempting to add 0 Fertilizer, it is now possible to add a self-referential node to the end FIFO list, but only if this is the first Fertilizer NFT to be minted for the current season by twice calling `FertilizerFacet.mintFertilizer(0, 0, 0, mode)`. The validation performed to prevent duplicate ids is erroneously bypassed given the Fertilizer amount for the given Id remains zero.

```
    function push(uint128 id) internal {
        AppStorage storage s = LibAppStorage.diamondStorage();
        if (s.fFirst == 0) {
            // Queue is empty
            s.season.fertilizing = true;
            s.fLast = id;
            s.fFirst = id;
        } else if (id <= s.fFirst) {
            // Add to front of queue
            setNext(id, s.fFirst);
            s.fFirst = id;
        } else if (id >= s.fLast) { // @audit this block is entered twice
            // Add to back of queue
            setNext(s.fLast, id); // @audit the second time, a reference is added to the same id
            s.fLast = id;
        } else {
            // Add to middle of queue
            uint128 prev = s.fFirst;
            uint128 next = getNext(prev);
            // Search for proper place in line
            while (id > next) {
                prev = next;
                next = getNext(next);
            }
            setNext(prev, id);
            setNext(id, next);
        }
    }
```

Despite first perhaps seeming harmless, this element can never be remove unless otherwise overridden:

```
    function pop() internal returns (bool) {
        AppStorage storage s = LibAppStorage.diamondStorage();
        uint128 first = s.fFirst;
        s.activeFertilizer = s.activeFertilizer.sub(getAmount(first)); // @audit getAmount(first) would
↪   return 0
        uint128 next = getNext(first);
        if (next == 0) { // @audit next != 0, therefore this conditional block is skipped
            // If all Unfertilized Beans have been fertilized, delete line.
            require(s.activeFertilizer == 0, "Still active fertilizer");
            s.fFirst = 0;
            s.fLast = 0;
            s.season.fertilizing = false;
            return false;
        }
        s.fFirst = getNext(first); // @audit this gets s.first again
        return true; // @audit always returns true for a self-referential node
    }
```

`LibFertilizer::pop` is used in `Sun::rewardToFertilizer` which is called through `Sun::rewardBeans` when fertilizing. This function is called through `Sun::stepSun` if the current Bean price is above peg. By preventing the last element from being popped from the list, assuming this element is reached, an infinite loop occurs given that the `while` loop continues to execute, resulting in denial-of-service on `SeasonFacet::gm` when above peg.

The most remarkable detail of this issue is that this state can be forced when above peg and having already been fully recapitalized. Given that it is not possible to mint additional Fertilizer with the associated Beans, this means that a DoS attack can be performed on `SeasonFacet::gm` once recapitalization is reached if the BEAN price is above peg.

Impact: It is possible to perform a denial-of-service (DoS) attack on `SeasonFacet::gm` if the Bean price is above the peg, either once fully recapitalized or when reaching the last element of the Fertilizer FIFO list.

Proof of Concept: This coded PoC can be run by:

1. Creating file `Beantalk/protocol/test/POCs/mint0Fertilizer.test.js`
2. Navigating to `Beantalk/protocol`
3. Running `yarn test --grep "DOS last fertilizer payment through minting 0 fertilizers"`

Recommended Mitigation: Despite being a complex issue to explain, the solution is as simple as replacing > with >= in `LibFertilizer::addFertilizer` as below:

```diff
    function addFertilizer(
        uint128 season,
        uint256 fertilizerAmount,
        uint256 minLP
    ) internal returns (uint128 id) {
        AppStorage storage s = LibAppStorage.diamondStorage();

        uint128 fertilizerAmount128 = fertilizerAmount.toUint128();

        // Calculate Beans Per Fertilizer and add to total owed
        uint128 bpf = getBpf(season);
        s.unfertilizedIndex = s.unfertilizedIndex.add(
            fertilizerAmount.mul(bpf)
        );
        // Get id
        id = s.bpf.add(bpf);
        // Update Total and Season supply
        s.fertilizer[id] = s.fertilizer[id].add(fertilizerAmount128);
        s.activeFertilizer = s.activeFertilizer.add(fertilizerAmount);
        // Add underlying to Unripe Beans and Unripe LP
        addUnderlying(fertilizerAmount.mul(DECIMALS), minLP);
        // If not first time adding Fertilizer with this id, return
-       if (s.fertilizer[id] > fertilizerAmount128) return id;
+       if (s.fertilizer[id] >= fertilizerAmount128) return id; // prevent infinite loop in
↪   `Sun::rewardToFertilizer` when attempting to add 0 Fertilizer, which could DoS `SeasonFacet::gm`
↪   when recapitalization is fulfilled
        // If first time, log end Beans Per Fertilizer and add to Season queue.
        push(id);
        emit SetFertilizer(id, bpf);
    }
```

Beanstalk Farms: Added a > 0 check to the `mintFertilizer` function in commit hash 4489cb8.

Cyfrin: Acknowledged. The Beanstalk Farms team has opted to add validation in `FertilizerFacet::mintFertilizer`. This alternative saves more gas compared to the one suggested; however, this issue should be considered in the future if `LibFertilizer::addFertilizer` is used anywhere else. This is the case in `FertilizerFacet::addFertilizerOwner` but assumedly will not be an issue as the owner would not send this type of transaction.

7.3 Low Risk

7.3.1 Incorrect handling of metadata traits in the attributes of `MetadataFacet::uri`

Description: For fully on-chain metadata, external clients expect the URI of a token to contain a base64 encoded JSON object that contains the metadata and base64 encoded SVG image. As raised previously, if these attributes are intented to be utilized as metadata traits then failure to correctly handle the packed encoding of the attributes variable as an array of JSON objects in `MetadataFacet::uri` results in non-standard JSON metadata when subsequently returned, meaning it cannot be fully utilized by external clients.

Impact: External clients such as OpenSea are currently unable to display Beanstalk token metadata traits due to non-standard JSON formatting.

Recommended Mitigation: Refactor the inline metadata attributes as an array of metadata trait objects, ensuring the resulting encoded bytes are that of valid JSON.

Beanstalk Farms: Fixed in commit 47fef03.

Cyfrin: Acknowledged.

7.4 Informational

7.4.1 Resetting of `withdrawSeasons` state was not executed on-chain as part of the BIP-36 upgrade

The addition of `s.season.withdrawSeasons = 0` to `InitBipNewSilo::init` does not appear to have been present in the version executed as part of the BIP-36 upgrade. Therefore, to have the state of Beanstalk accurately reflect this change, another upgrade should be performed to have this logic executed on-chain.

Beanstalk Farms: Fixed in commit cca6250.

Cyfrin: Acknowledged.

7.4.2 Changes to initialization contracts are not recommended after they are executed on-chain

It is understood that certain modifications to BIP initialization contracts have been made retroactively with the intention that, if run again, any new deployments of the Beanstalk protocol by replaying this history will reflect the current state of Beanstalk. One particular modification to `InitDiamond::init`, setting the `stemStartSeason` to zero, while seemingly benign as migration logic in `LibSilo` appears to be bypassed, would result in underdlow within `LibLegacyTokenSilo::_calcGrownStalkForDeposit` when calculating the Season diff. This issue will be present until `InitBipNewSilo::init` excutes, setting the `stemStartSeason` state to the Season in which it is executed. It is therefore recommended that initialization scripts are ossified after being executed on-chain to maintain an accurate history of the protocol: its mechanism developments, bugs and related upgrades/mitigations.

Beanstalk Farms: The purpose of such changes is to future proof future deployments of Beanstalk. If someone were to deploy a fresh Beanstalk, it is important that the protocol continues to function as expected with all upgrades already implemented.

The expectation is that a new Beanstalk would be initialized only with `InitDiamond` and that Beanstalk would automatically be on the newest version. The other `Init` contracts are intended strictly to migrate from a previous version to the next.

The `LibLegacyTokenSilo` is only used to provide legacy support and migration functionality for Silo V2. This includes, the `MigrationFacet`, `LegacyClaimWithdrawalFacet` and `seasonToStem(address token, uint32 season)` in `SiloExit`. The expectation is that a new Beanstalk would be deployed immediately with the Silo V3 upgrade and thus have no reason to be backwards compatable with Silo V2 or support migration from V2 to V3 in any capacity.

Cyfrin: Acknowledged.

7.4.3 Lack of slippage protection when removing liquidity from BEAN:3CRV MetaPool and adding liquidity to BEAN:ETH Well could result in loss of funds due to sandwich attack

Currently, the second and third steps of the *Migration Process*, as provided in BIP-38 specification, are not included in the scope of this BIP. The primary risk associated with these steps is the swap of BEAN:3CRV LP Tokens for BEAN:ETH Well LP Tokens, given the size of the swap to be performed. Sandwiching of the transaction that executes this swap could result in loss of funds. Therefore, the use of reasonable slippage parameters is essential to prevent this. It is understood that the current use of zero slippage parameters within the `beanEthMigration.js` migration script when removing liquidity from the MetaPool and adding liquidity to the Well is only intended for testing purposes. The swap path from 3CRV -> WETH will either be executed manually via the BCM on a DEX aggregator with MEV protection or via an OTC swap, and the BCM will ensure the proper use of slippage parameters when removing/adding liquidity. It is essential that this is the case.

Beanstalk Farms: This script is only expected to be used to mock the migration to aid in testing. The expectation is that it will never be used to execute code on mainnet and thus no slippage parameter is added.

Cyfrin: Acknowledged.

7.4.4 `LibEthUsdOracle::getEthUsdPrice` design changes should be documented

Before BIP-38, the `LibEthUsdOracle::getEthUsdPrice` function had the following behavior:

1. If the difference between the Chainlink ETH/USD oracle and the Uniswap ETH/USDC TWAP oracle (considering a 15-minute window) prices was below `0.5%`, then it would return the average of both values. Now, this difference should be below `0.3%`.
2. If the difference between the Chainlink ETH/USD oracle and the Uniswap ETH/USDC TWAP oracle (considering a 15-minute window) was greater than the difference between the Chainlink ETH/USD oracle and the Uniswap ETH/USDT TWAP oracle (considering a 15-minute window), then:
    - If the difference between the Chainlink ETH/USD oracle and the Uniswap ETH/USDT TWAP oracle (considering a 15 minute-window) prices was below `2%`, it would return the average of these two prices. Now, this difference should be less than `1%`.
    - Otherwise, it would return 0, indicating that the oracle is broken or stale. Now, it returns the Chainlink ETH/USD oracle price, assuming it is correct.
3. Otherwise:
    - If the difference between the Chainlink ETH/USD oracle and the Uniswap ETH/USDC TWAP oracle (considering a 15-minute window) prices was below `2%`, it would return the average of these two prices. Now, this difference should be less than `1%`.
    - Otherwise, it would return 0, indicating that the oracle is broken or stale. Now, it returns the Chainlink ETH/USD oracle price, assuming it is correct.

In essence, this function now assumes that the Chainlink ETH/USD price is correct as long as it is not stale or broken (if it returns 0). In cases where the difference between this price and the Uniswap ETH/USDC TWAP oracle price or Uniswap ETH/USDT TWAP oracle price is outside certain thresholds, it considers and averages with one of these values. Previously, if this difference was not within certain bounds, the oracle was considered to be broken.

Beanstalk Farms: This change was actually made before BIP-37 was deployed, but this modification was omitted from the previous Cyfrin audit. Thus, no functionality in `getEthUsdPrice` changed as a part of BIP-38.

The comments in `LibEthUsdOracle` were not correct and have been updated in commit 968f783.

Cyfrin: Acknowledged. Comments now match the code's intention.

7.4.5 Logic in `LibFertilizer::push` related to (deprecated) intermediate addition of Fertilizer to FIFO list should be removed

Intermediate addition to the FIFO list was formerly allowed only by the Beanstalk DAO, but this functionality has since been deprecated in the current upgrade with the removal of `FertilizerFacet::addFertilizerOwner`. Consequently, the corresponding logic in `LibFertilizer::push` should be removed as this now represents unreachable code.

Beanstalk Farms: the `push(...)` function is still used internally here.

The highlighted segment is not used reachable anymore, but in the case where the humidity is changed for some reason, it could again be reached. For this reason, the decision was made to leave it in.

Cyfrin: Acknowledged.

7.4.6 Consider moving the `MetadataFacet::uri` disclaimer from metadata attributes to the description

The disclaimer within `MetadataFacet::uri` currently resides at the end of the JSON attributes; however, this may be better placed within the metadata description instead.

Beanstalk Farms: The disclaimer placement was largely inspired by Uniswap V3's NFT and thus, feel that the attribute section is an adequate place to keep it.

Cyfrin: Acknowledged.

7.4.7 Incorrect comment in `MetadataImage::sciNotation` should be corrected

`MetadataImage::sciNotation` is intended to convert an input Stem to its string representation, using scientific notation if the value is greater than 1e5. Related comments referencing 1e7 as the threshold are incorrect and so should be modified to 1e5.

Beanstalk Farms: Fixed in commit 81e452e.

Cyfrin: Acknowledged.

7.4.8 Continued reference to "Seeds" in `InitBipBasinIntegration::init` is confusing

With the deprecation of the "Seeds" terminology, continued reference is confusing and all instances should be updated to instead refer to the earned Stalk per BDV per Season.

Beanstalk Farms: Updated names in commit ba1d42b.

7.4.9 `InitBipBasinIntegration` NatSpec title tag is inconsistent with the file/contract name

The title tag of the `InitBipBasinIntegration` NatSpec is inconsistent with the file/contract name and should be updated to match.

Beanstalk Farms: Fixed in commit c03f635.

Cyfrin: Acknowledged.

7.4.10 Conditional block in `WellPrice::getConstantProductWell` can be removed

The else block in `WellPrice::getConstantProductWell`, which handles the case when it is not possible to determine a price for Bean, is not necessary and can be removed as the default value of the `pool.price` is already zero.

Beanstalk Farms: Removed in commit 8aae31d.

Cyfrin: Acknowledged.

7.4.11 Unsafe cast in `WellPrice::getDeltaB`

While not likely to overflow, there is an unsafe cast in `WellPrice::getDeltaB` which could be replaced with a safe cast.

Beanstalk Farms: Fixed in commit ff742a6.

7.4.12 Typo in `FertilizerFacet::getMintFertilizerOut` NatSpec

The NatSpec of `FertilizerFacet::getMintFertilizerOut` currently refers to Fertilizer as `Fertilize` which should be corrected.

Beanstalk Farms: Fixed in commit 373c094.

Cyfrin: Acknowledged.

7.4.13 Typo in comment within `LibSilo::_mow`

The following typo in `LibSilo::_mow` should be corrected:

```
- //sop stuff only needs to be updated once per season
- //if it started raininga nd it's still raining, or there was a sop
+ // sop stuff only needs to be updated once per season
+ // if it started raining and it's still raining, or there was a sop
```

Beanstalk Farms: Fixed in commit d27567c.

Cyfrin: Acknowledged.