// HALBORN

Beanstalk - Basin Integration Upgrade

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **May 4th, 2023 - June 8th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	06/02/2023	Miguel Jalon
0.2	Document Edits	06/08/2023	Miguel Jalon
0.3	Draft Review	06/09/2023	Francisco González
0.4	Draft Review	06/09/2023	Grzegorz Trawinski
0.5	Draft Review	06/09/2023	Piotr Cielas
0.6	Draft Review	06/14/2023	Gabi Urrutia
1.0	Remediation Plan	07/17/2023	Roberto Reigada
1.1	Remediation Plan Review	07/18/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Miguel Jalon	Halborn	Miguel.Jalon@halborn.com
Francisco González	Halborn	Francisco.Villarejo@halborn.com
Grzegorz Trawinski	Halborn	Grzegorz.Trawinski@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Beanstalk is a stablecoin protocol where BEAN is the main asset around which the whole Beanstalk farm works. The users interact with the farm by using the Beanstalk services, allowing BEAN to periodically cross the peg.

The Beanstalk Basin Integration aims to integrate the different Wells deployed in the Basin to allow Stalkholders (DAO members) to interact with the Wells and swap (or convert) assets that have been deposited without having to make a withdrawal and thus losing all of the grown and accumulated stalk.

Beanstalk engaged Halborn to conduct a security assessment on their smart contracts beginning on May 4th, 2023 and ending on June 8th, 2023. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided five weeks for the engagement and assigned a full-time security engineer to assessment the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some code inefficiencies that were addressed/acknowledged by the Beanstalk team.

1.3 SCOPE

1. Beanstalk Protocol:

- Commit ID: b28a58d134fb9a53e1a30e9df695ffbd28ecaf2f
- Smart Contracts in scope:
    1. C.sol
    2. AppStorage.sol
    3. InitWhitelist.sol
    4. BDVFacet.sol
    5. ConvertFacet.sol
    6. EnrootFacet.sol
    7. SiloFacet.sol
    8. WhitelistFacet.sol
    9. Oracle.sol
    10. SeasonFacet.sol
    11. LibConvert.sol
    12. LibConvertData.sol
    13. LibUnripeConvert.sol
    14. LibWellConvert.sol
    15. LibMinting.sol
    16. LibWellMinting.sol
    17. LibTokenSilo.sol
    18. LibWhitelist.sol
    19. LibWell.sol
    20. LibWellBdv.sol

2. Basin (Wells):

- Commit ID: 0949cd7d9658a0525c526b9c771a442c65ee204a
- Smart Contracts in scope:
    1. Well.sol
    2. ConstantProduct.sol
    3. ConstantProduct2.sol
    4. LibMath.sol
    5. GeoEmaAndCumSumSmaPump.sol

Fixed Commit ID: 78d7045a4e6900dfbdc5f1119b202b4f30ff6ab8

1.4 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the contracts' solidity code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the assessment:

- Smart contract manual code review and walkthrough.
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing with custom scripts. (Foundry).
- Static Analysis of security for scoped contract, and imported functions manually.
- Scanning of solidity files for vulnerabilities, security hotspots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)

- Testnet deployment (Anvil).

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	3

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) ENCODE TYPE NOT ENFORCED WHILE WHITELISTING A TOKEN	Informational (1.9)	ACKNOWLEDGED
(HAL-02) UNNECESARY ELSE STATEMENT AND DELTAB VARIABLE INITIALIZATION	Informational (1.9)	SOLVED - 07/17/2023
(HAL-03) GAS INEFFICIENCY: > 0 IN A UINT256 INSTEAD OF != 0	Informational (1.9)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) ENCODE TYPE NOT ENFORCED WHILE WHITELISTING A TOKEN - INFORMATIONAL (1.9)

Description:

When whitelisting a token with the `whitelistTokenWithEncodeType()` function, the `encodeType` passed by parameter is not checked if it is `0x00` or `0x01`. Then, if the `beanDenominatedValue()` function is called (which is called every time the `bdv` has to be calculated, and it is required for most of the use cases of a token), the `tx` reverts if the token has an invalid `encodeType`. The inconvenience is that the token cannot be used in the protocol with this incorrectly set parameter.

Code Location:

Listing 1: WhitelistFacet.sol

```
96          function whitelistTokenWithEncodeType(
97          address token,
98          bytes4 selector,
99          uint32 stalkIssuedPerBdv,
100         uint32 stalkEarnedPerSeason,
101         bytes1 encodeType
102     ) external payable {
103         LibDiamond.enforceIsOwnerOrContract();
104         LibWhitelist.whitelistToken(
105             token,
106             selector,
107             stalkIssuedPerBdv,
108             stalkEarnedPerSeason,
109             encodeType
110         );
111     }
```

Listing 2: LibTokenSilo.sol (Line 295)

```
274     function beanDenominatedValue(address token, uint256 amount)
275         internal
276         view
277         returns (uint256 bdv)
278     {
279         AppStorage storage s = LibAppStorage.diamondStorage();
280         require(s.ss[token].selector != bytes4(0), "Silo: Token
 ↳ not whitelisted");
281
282         bytes memory callData;
283         if (s.ss[token].encodeType == 0x00) {
284             callData = abi.encodeWithSelector(
285                 s.ss[token].selector,
286                 amount
287             );
288         } else if (s.ss[token].encodeType == 0x01) {
289             callData = abi.encodeWithSelector(
290                 s.ss[token].selector,
291                 token,
292                 amount
293             );
294         } else {
295             revert("Silo: Invalid encodeType");
296         }
```

Proof of Concept:

1. Beanstalk whitelists a well LP token into the protocol
2. A user adds liquidity to the well
3. The user tries to deposit into the Silo the LP tokens previously received
4. The transaction reverts with Silo: Invalid encodeType

Listing 3: HalbornBeanstalkTest.t.sol (Line 8)

```
1     function test_vuln_CORE_Whitelist_006() public {
2         vm.prank(owner);
3         whitelistFacet.whitelistTokenWithEncodeType(wellAdd,
↳ bytes4(keccak256("wellBdv(address,uint256)")), uint32(10000),
↳ uint32(1), 0x02);
4
5         xAddLiqSimple(alice, bean.balanceOf(alice), weth.balanceOf
↳ (alice));
6         vm.startPrank(alice);
7         well.approve(address(siloFacet), well.balanceOf(alice));
8         siloFacet.deposit(wellAdd, well.balanceOf(alice),
↳ LibTransfer.From.EXTERNAL);
9     }
```

```
Running 1 test for test/foundry/HalbornBeanstalkTest.t.sol:HalbornBeanstalkTest
[FAIL. Reason: Silo: Invalid encodeType] test_vuln_CORE_Whitelist_006() (gas: 417632)
Logs:
  TX: ADDING LIQUIDITY...  0x61A1D7fD8C9bbd82932D99DFD47bD2581C23b08c

Test result: FAILED. 0 passed; 1 failed; finished in 636.99ms
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:H/D:N/Y:N/R:F/S:U (1.9)

Recommendation:

Add a require assertion enforcing the encodeType to be 0x00 or 0x01.

Remediation Plan:

ACKNOWLEDGED: The Beanstalk team acknowledged this finding.

4.2 (HAL-02) UNNECESARY ELSE STATEMENT AND DELTAB VARIABLE INITIALIZATION - INFORMATIONAL (1.9)

Description:

In the EVM, all variables are initially zeros. The LibWellMinting.check() function does not need to initialize the deltaB variable in the else condition as it is already defined in the return variables of the function, and is already set to zero.

Code Location:

Listing 4: LibWellMinting.sol (Line 72)

```
51     function check(
52         address well
53     ) internal view returns (int256 deltaB) {
54         bytes memory lastSnapshot = LibAppStorage
55             .diamondStorage()
56             .wellOracleSnapshots[well];
57         // If the length of the stored Snapshot for a given Well
↳ is 0,
58         // then the Oracle is not initialized.
59         if (lastSnapshot.length > 0) {
60             (deltaB, ) = twaDeltaB(well, lastSnapshot);
61         } else {
62             deltaB = 0;
63         }
64         deltaB = LibMinting.checkForMaxDeltaB(deltaB);
65     }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:H/D:N/Y:N/R:F/S:U (1.9)

Recommendation:

Consider removing the else condition from the LibWellMinting.check() function:

Listing 5: LibWellMinting.sol

```
51     function check(
52         address well
53     ) internal view returns (int256 deltaB) {
54         bytes memory lastSnapshot = LibAppStorage
55             .diamondStorage()
56             .wellOracleSnapshots[well];
57         // If the length of the stored Snapshot for a given Well
↳ is 0,
58         // then the Oracle is not initialized.
59         if (lastSnapshot.length > 0) {
60             (deltaB, ) = twaDeltaB(well, lastSnapshot);
61         }
62
63         deltaB = LibMinting.checkForMaxDeltaB(deltaB);
64     }
```

Remediation Plan:

SOLVED: The Beanstalk team solved this issue.

Commit ID : 78d7045a4e6900dfbdc5f1119b202b4f30ff6ab8.

4.3 (HAL-03) GAS INEFFICIENCY: > 0 IN A UINT256 INSTEAD OF != 0 - INFORMATIONAL (1.9)

Description:

In the ConvertFacet contract, the _depositTokensForConvert() function uses > 0 to compare if it's different from 0 instead of using != 0 which is more gas efficient when used with unsigned integer data types.

Code Location:

Listing 6: ConvertFacet.sol (Line 202)

```
196     function _depositTokensForConvert(
197         address token,
198         uint256 amount,
199         uint256 bdv,
200         uint256 grownStalk // stalk grown previously by this
 ↳ deposit
201     ) internal returns (int96 stem) {
202         require(bdv > 0 && amount > 0, "Convert: BDV or amount is
 ↳ 0.");
```

Proof of Concept:

Listing 7: HalbornPoC.sol (Lines 9,14)

```
 4 contract HalbornPoC {
 5
 6     uint256 x;
 7     function printGreater() public {
 8         x = 1;
 9         if (x > 0) {}
10     }
11
12     function printDifferent() public {
```

```
13          x = 1;
14          if (x != 0) {}
15      }
16  }
```

- Using printGreater()

gas	49795 gas
transaction cost	43300 gas
execution cost	22236 gas

- Using printDifferent()

gas	26936 gas
transaction cost	23422 gas
execution cost	2358 gas

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:H/D:N/Y:N/R:F/S:U (1.9)

Recommendation:

It is recommended to use `!= 0` instead of `> 0` to compare `uint` variables.

Listing 8: ConvertFacet.sol (Line 202)

```
196     function _depositTokensForConvert(
197         address token,
198         uint256 amount,
199         uint256 bdv,
200         uint256 grownStalk // stalk grown previously by this
 ↳ deposit
201     ) internal returns (int96 stem) {
202         require(bdv != 0 && amount != 0, "Convert: BDV or amount
 ↳ is 0.");
```

Remediation Plan:

ACKNOWLEDGED: The Beanstalk team acknowledged the issue.

MANUAL TESTING

MANUAL TESTING

For the integration of the Wells in the Beanstalk protocol, the next steps had to be performed:

1. Fork the mainnet running it locally by using Anvil
2. Deploy a Well with a deployment script using the actual GeoEmaAndCumSmaPump pump contract in the forked environment
3. Update the facets by:
    a) Removing with Diamond Cut all the external functions -or the external functions whose internal functions or libraries- have changed or have been removed from the mainnet in the new commits
    b) Adding with Diamond Cut all the new external functions -or the external functions whose internal functions or libraries- has changed or has been added from the mainnet in the new commits
4. Use Foundry for the Unit and Integration testing connected to the Anvil with the mainnet state and a Well deployed

The assessment mainly focused on the following points:

ENCODE TYPE (test_CORE_Whitelist):

- Making sure the upgrades of the Silo Whitelist are correct, focusing on the whitelist() function the encodeType parameter that allows 0x01 type for the wellBdv function.

WELL BDV (test_CORE_WellBdv):

- Tracking that the wellBdv() function is correctly implemented when a Well LP token has been previously whitelisted into Beanstalk.

CONVERTS (test_CORE_Convert):

- Checking that the convertLPToBeans() and convertBeansToLP() functions work as intended in the LibWellConvert library also taking into account the new two convert types BEANS_TO_WELL_LP, WELL_LP_TO_BEANS

TWA DELTA B (test_CORETwaDBOracle):

- Checking the Oracle.sol functionality to get the correct TWA Delta B in any well containing the BEAN token. Furthermore, checking stepOracle() and totalDeltaB() functions in the Oracle contract.

ENROOT DEPOSIT (test_CORE_Enroot):

- Checking the enrootDeposit() functions are correctly migrated to the new facet EnrootFacet.sol.

CONSTANT PRODUCT FUNCTIONS (test_WELL):

- Checking that the new changes made within ConstantProduct.sol and ConstantProduct2.sol WellFunction contracts are correctly implemented.

MANUAL TESTING

```
Running 58 tests for test/foundry/HalbornBeanstalkTest.t.sol:HalbornBeanstalkTest
[PASS] test_CORE_Convert_000() (gas: 243212)
[PASS] test_CORE_Convert_001() (gas: 673215)
[PASS] test_CORE_Convert_010() (gas: 467974)
[PASS] test_CORE_Convert_011() (gas: 428172)
[PASS] test_CORE_Convert_012() (gas: 428718)
[PASS] test_CORE_Convert_013() (gas: 468211)
[PASS] test_CORE_Convert_015() (gas: 888854)
[PASS] test_CORE_Convert_016() (gas: 616537)
[PASS] test_CORE_Convert_020() (gas: 343921)
[PASS] test_CORE_Convert_021() (gas: 431159)
[PASS] test_CORE_Convert_022() (gas: 947468)
[PASS] test_CORE_Enroot_000() (gas: 760142)
[PASS] test_CORE_Enroot_001() (gas: 1108539)
[PASS] test_CORE_TwaDBOracle_000() (gas: 477642)
[PASS] test_CORE_TwaDBOracle_001() (gas: 445146)
[PASS] test_CORE_TwaDBOracle_002() (gas: 430537)
[PASS] test_CORE_TwaDBOracle_003() (gas: 431753)
[PASS] test_CORE_TwaDBOracle_004() (gas: 445002)
[PASS] test_CORE_TwaDBOracle_010() (gas: 525185)
[PASS] test_CORE_TwaDBOracle_011() (gas: 483359)
[PASS] test_CORE_TwaDBOracle_012() (gas: 550061)
[PASS] test_CORE_TwaDBOracle_300() (gas: 803155)
[PASS] test_CORE_WellBdv_001() (gas: 493716)
[PASS] test_CORE_WellBdv_002() (gas: 971975)
[PASS] test_CORE_WellBdv_012() (gas: 757456)
[PASS] test_CORE_WellBdv_013() (gas: 747444)
[PASS] test_CORE_WellBdv_014() (gas: 590910)
[PASS] test_CORE_WellBdv_020() (gas: 500080)
[PASS] test_CORE_WellBdv_127() (gas: 383140)
[PASS] test_CORE_WellBdv_128() (gas: 606815)
[PASS] test_CORE_Whitelist_001() (gas: 48719)
[PASS] test_CORE_Whitelist_002() (gas: 55279)
[PASS] test_CORE_Whitelist_003() (gas: 55256)
[PASS] test_CORE_Whitelist_004() (gas: 123198)
[PASS] test_CORE_Whitelist_005() (gas: 31705)
[PASS] test_CORE_Whitelist_007() (gas: 827238)
[PASS] test_CORE_Whitelist_008() (gas: 123263)
[PASS] test_CORE_Whitelist_010() (gas: 123241)
[PASS] test_CORE_Whitelist_011() (gas: 303599)
[PASS] test_CORE_Whitelist_020() (gas: 292181)
[PASS] test_CORE_Whitelist_021() (gas: 466374)
[PASS] test_WELL_000() (gas: 224536)
[PASS] test_WELL_001() (gas: 383247)
[PASS] test_WELL_002() (gas: 383233)
[PASS] test_WELL_003() (gas: 372350)
[PASS] test_WELL_004() (gas: 446547)
[PASS] test_WELL_005() (gas: 542940)
[PASS] test_WELL_009() (gas: 345376)
[PASS] test_WELL_010() (gas: 345419)
[PASS] test_WELL_011() (gas: 345417)
[PASS] test_WELL_100() (gas: 383224)
[PASS] test_WELL_101() (gas: 224537)
[PASS] test_WELL_102() (gas: 383235)
[PASS] test_WELL_111(uint256) (runs: 256, μ: 446669, ~: 446669)
[PASS] test_WELL_112(uint256) (runs: 256, μ: 648949, ~: 648970)
[PASS] test_WELL_113(uint256) (runs: 256, μ: 817440, ~: 817455)
[PASS] test_X_deposit() (gas: 200940)
[FAIL. Reason: Silo: Invalid encodeType] test_hal01_CORE_Whitelist_006() (gas: 417612)
Test result: FAILED. 57 passed; 1 failed; finished in 790.59ms
```

AUTOMATED TESTING

6.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats, Slither was run on the all-scoped contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' ABIs across the entire code-base.

Slither Results:

BDVFacet.sol

```
LibCurve.getY(uint256,uint256[2],uint256,uint256) (contracts/libraries/Curve/LibCurve.sol#49-81) performs a multiplication on the result of a division:
        -c = (c * D) / (_x * N_COINS) (contracts/libraries/Curve/LibCurve.sol#67)
        -c = (c * D * A_PRECISION) / (Ann * N_COINS) (contracts/libraries/Curve/LibCurve.sol#70)
LibCurve.getD(uint256[2],uint256) (contracts/libraries/Curve/LibCurve.sol#83-112) performs a multiplication on the result of a division:
        -D_P = (D_P * D) / (xp[_j] * N_COINS) (contracts/libraries/Curve/LibCurve.sol#101)
        -D = (((Ann * S) / A_PRECISION + D_P * N_COINS) * D) / (((Ann - A_PRECISION) * D) / A_PRECISION + (N_COINS + 1) * D_P) (contracts/libraries/Curve/LibCurve.sol#104-106)
LibCurve.getYD(uint256,uint256,uint256[2],uint256) (contracts/libraries/Curve/LibCurve.sol#114-145) performs a multiplication on the result of a division:
        -c = (c * D) / (_x * N_COINS) (contracts/libraries/Curve/LibCurve.sol#131)
        -c = (c * D * A_PRECISION) / (Ann * N_COINS) (contracts/libraries/Curve/LibCurve.sol#134)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

LibCurve.getY(uint256,uint256[2],uint256,uint256)._i_scope_0 (contracts/libraries/Curve/LibCurve.sol#74) is a local variable never initialized
LibCurve.getYD(uint256,uint256,uint256[2],uint256)._i_scope_0 (contracts/libraries/Curve/LibCurve.sol#138) is a local variable never initialized
LibCurve.getD(uint256[2],uint256)._j (contracts/libraries/Curve/LibCurve.sol#100) is a local variable never initialized
LibCurve.getD(uint256[2],uint256)._i (contracts/libraries/Curve/LibCurve.sol#91) is a local variable never initialized
LibCurve.getD(uint256[2],uint256)._i_scope_0 (contracts/libraries/Curve/LibCurve.sol#98) is a local variable never initialized
LibCurve.getYD(uint256,uint256,uint256[2],uint256)._i (contracts/libraries/Curve/LibCurve.sol#127) is a local variable never initialized
LibCurve.getY(uint256,uint256[2],uint256,uint256)._i (contracts/libraries/Curve/LibCurve.sol#62) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

LibAppStorage.diamondStorage() (contracts/libraries/LibAppStorage.sol#15-19) uses assembly
        - INLINE ASM (contracts/libraries/LibAppStorage.sol#16-18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['=0.7.6', '>=0.6.0<0.8.0']
        - >=0.6.0<0.8.0 (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#3)
        - >=0.6.0<0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#3)
        - >=0.6.0<0.8.0 (lib/openzeppelin-contracts/contracts/utils/Counters.sol#3)
        - =0.7.6 (contracts/C.sol#3)
        - ABIEncoderV2 (contracts/C.sol#4)
        - =0.7.6 (contracts/beanstalk/AppStorage.sol#3)
        - ABIEncoderV2 (contracts/beanstalk/AppStorage.sol#4)
        - =0.7.6 (contracts/beanstalk/silo/BDVFacet.sol#5)
        - ABIEncoderV2 (contracts/beanstalk/silo/BDVFacet.sol#6)
        - =0.7.6 (contracts/interfaces/IBean.sol#3)
        - ABIEncoderV2 (contracts/interfaces/IBean.sol#4)
        - ABIEncoderV2 (contracts/interfaces/ICurve.sol#2)
        - =0.7.6 (contracts/interfaces/ICurve.sol#3)
        - ABIEncoderV2 (contracts/interfaces/IDiamondCut.sol#2)
        - =0.7.6 (contracts/interfaces/IDiamondCut.sol#3)
        - ABIEncoderV2 (contracts/interfaces/IFertilizer.sol#2)
        - =0.7.6 (contracts/interfaces/IFertilizer.sol#3)
        - ABIEncoderV2 (contracts/interfaces/IProxyAdmin.sol#2)
        - =0.7.6 (contracts/interfaces/IProxyAdmin.sol#3)
        - =0.7.6 (contracts/libraries/Curve/LibBeanMetaCurve.sol#3)
        - ABIEncoderV2 (contracts/libraries/Curve/LibBeanMetaCurve.sol#4)
        - =0.7.6 (contracts/libraries/Curve/LibCurve.sol#3)
        - ABIEncoderV2 (contracts/libraries/Curve/LibCurve.sol#4)
        - =0.7.6 (contracts/libraries/Curve/LibMetaCurve.sol#3)
        - ABIEncoderV2 (contracts/libraries/Curve/LibMetaCurve.sol#4)
        - =0.7.6 (contracts/libraries/Decimal.sol#5)
        - ABIEncoderV2 (contracts/libraries/Decimal.sol#6)
        - =0.7.6 (contracts/libraries/LibAppStorage.sol#3)
        - ABIEncoderV2 (contracts/libraries/LibAppStorage.sol#4)
        - =0.7.6 (contracts/libraries/LibUnripe.sol#3)
        - ABIEncoderV2 (contracts/libraries/LibUnripe.sol#4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

C.UniV3EthUsdc() (contracts/C.sol#179-181) is never used and should be removed
C.bean() (contracts/C.sol#147-149) is never used and should be removed
C.beanAddress() (contracts/C.sol#103-105) is never used and should be removed
C.curve3PoolAddress() (contracts/C.sol#171-173) is never used and should be removed
C.curveMetapoolAddress() (contracts/C.sol#107-109) is never used and should be removed
C.curveZap() (contracts/C.sol#163-165) is never used and should be removed
C.curveZapAddress() (contracts/C.sol#167-169) is never used and should be removed
C.dollarPerUnripeLP() (contracts/C.sol#207-209) is never used and should be removed
C.exploitAddLPRatio() (contracts/C.sol#211-213) is never used and should be removed
C.fertilizer() (contracts/C.sol#183-185) is never used and should be removed
C.fertilizerAddress() (contracts/C.sol#187-189) is never used and should be removed
C.fertilizerAdmin() (contracts/C.sol#191-193) is never used and should be removed
C.getBlockLengthSeconds() (contracts/C.sol#79-81) is never used and should be removed
C.getChainId() (contracts/C.sol#83-85) is never used and should be removed
```

ConvertFacet.sol

```
ConvertFacet.enrootDeposits(address,int96[],uint256[]) (contracts/beanstalk/silo/ConvertFacet.sol#179-222) performs a multiplication on the result of a division:
        -bdv = amounts[i].mul(newBDV).div(ar.tokensRemoved) (contracts/beanstalk/silo/ConvertFacet.sol#196)
        -newStalk = newStalk.add(bdv.mul(_stalkPerBdv).add(LibSilo.stalkReward(stems[i],_lastStem,uint128(bdv)))) (contracts/beanstalk/silo/ConvertFacet.sol#206-214)
LibCurve.getY(uint256,uint256[2],uint256,uint256) (contracts/libraries/Curve/LibCurve.sol#49-81) performs a multiplication on the result of a division:
        -c = (c * D) / (_x * N_COINS) (contracts/libraries/Curve/LibCurve.sol#67)
        -c = (c * D * A_PRECISION) / (Ann * N_COINS) (contracts/libraries/Curve/LibCurve.sol#70)
LibCurve.getD(uint256[2],uint256) (contracts/libraries/Curve/LibCurve.sol#83-112) performs a multiplication on the result of a division:
        -D_P = (D_P * D) / (xp[_j] * N_COINS) (contracts/libraries/Curve/LibCurve.sol#101)
        -D = (((Ann * S) / A_PRECISION + D_P * N_COINS) * D) / (((Ann - A_PRECISION) * D) / A_PRECISION + (N_COINS + 1) * D_P) (contracts/libraries/Curve/LibCurve.sol#104-106)
LibCurve.getYD(uint256,uint256,uint256[2],uint256) (contracts/libraries/Curve/LibCurve.sol#114-145) performs a multiplication on the result of a division:
        -c = (c * D) / (_x * N_COINS) (contracts/libraries/Curve/LibCurve.sol#131)
        -c = (c * D * A_PRECISION) / (Ann * N_COINS) (contracts/libraries/Curve/LibCurve.sol#134)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse = (3 * denominator) ^ 2 (contracts/libraries/LibPRBMath.sol#246)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#250)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#251)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#252)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#253)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#254)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#255)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -prod0 = prod0 / twos (contracts/libraries/LibPRBMath.sol#234)
        -result = prod0 * inverse (contracts/libraries/LibPRBMath.sol#261)
LibUnripeSilo._removeUnripeLPDeposit(address,uint32,uint256) (contracts/libraries/Silo/LibUnripeSilo.sol#212-289) performs a multiplication on the result of a division:
        -removed = amount.mul(bdv1).div(amount1) (contracts/libraries/Silo/LibUnripeSilo.sol#230)
        -s.a[account].lp.depositSeeds[season] = s.a[account].lp.depositSeeds[season].sub(removed.mul(4)) (contracts/libraries/Silo/LibUnripeSilo.sol#231-235)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

Reentrancy in Silo._claimPlenty(address) (contracts/beanstalk/silo/SiloFacet/Silo.sol#147-154):
        External calls:
        - C.threeCrv().safeTransfer(account,plenty) (contracts/beanstalk/silo/SiloFacet/Silo.sol#150)
        State variables written after the call(s):
        - delete s.a[account].sop.plenty (contracts/beanstalk/silo/SiloFacet/Silo.sol#151)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

LibCurve.getD(uint256[2],uint256)._i_scope_0 (contracts/libraries/Curve/LibCurve.sol#98) is a local variable never initialized
LibLegacyTokenSilo._mowAndMigrate(address,address[],uint32[][],uint256[][]).perTokenData (contracts/libraries/Silo/LibLegacyTokenSilo.sol#371) is a local variable never initialized
LibCurve.getYD(uint256,uint256,uint256[2],uint256)._i (contracts/libraries/Curve/LibCurve.sol#127) is a local variable never initialized
LibCurve.getY(uint256,uint256[2],uint256,uint256)._i (contracts/libraries/Curve/LibCurve.sol#62) is a local variable never initialized
LibCurve.getD(uint256[2],uint256)._i (contracts/libraries/Curve/LibCurve.sol#91) is a local variable never initialized
LibDiamond.removeFunctions(address,bytes4[]).selectorIndex (contracts/libraries/LibDiamond.sol#162) is a local variable never initialized
LibSilo._removeDepositsFromAccount(address,address,int96[],uint256[]).i (contracts/libraries/Silo/LibSilo.sol#555) is a local variable never initialized
ConvertFacet.enrootDeposits(address,int96[],uint256[]).i (contracts/beanstalk/silo/ConvertFacet.sol#195) is a local variable never initialized
LibDiamond.replaceFunctions(address,bytes4[]).selectorIndex (contracts/libraries/LibDiamond.sol#147) is a local variable never initialized
LibCurve.getYD(uint256,uint256,uint256[2],uint256)._i_scope_0 (contracts/libraries/Curve/LibCurve.sol#138) is a local variable never initialized
ConvertFacet._withdrawTokens(address,int96[],uint256[],uint256).i (contracts/beanstalk/silo/ConvertFacet.sol#234) is a local variable never initialized
LibDiamond.addFunctions(address,bytes4[]).selectorIndex (contracts/libraries/LibDiamond.sol#129) is a local variable never initialized
LibLegacyTokenSilo._claimWithdrawals(address,address,uint32[]).i (contracts/libraries/Silo/LibLegacyTokenSilo.sol#587) is a local variable never initialized
LibLegacyTokenSilo._mowAndMigrate(address,address[],uint32[][],uint256[][]).migrateData (contracts/libraries/Silo/LibLegacyTokenSilo.sol#367) is a local variable never initialized
LibCurve.getD(uint256[2],uint256)._i (contracts/libraries/Curve/LibCurve.sol#109) is a local variable never initialized
LibDiamond.diamondCut(IDiamondCut.FacetCut[],address,bytes).facetIndex (contracts/libraries/LibDiamond.sol#104) is a local variable never initialized
LibCurve.getY(uint256,uint256[2],uint256,uint256)._i_scope_0 (contracts/libraries/Curve/LibCurve.sol#74) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

LibUnripeSilo.unripeLPDeposit(address,uint32).bdv (contracts/libraries/Silo/LibUnripeSilo.sol#306) is written in both
        (amount,bdv) = getBeanEthUnripeLP(account,season) (contracts/libraries/Silo/LibUnripeSilo.sol#312)
        bdv = uint256(s.a[account].legacyDeposits[C.unripeLPAddress()][season].bdv).add(legBdv) (contracts/libraries/Silo/LibUnripeSilo.sol#329-331)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#write-after-write
```

WhitelistFacet.sol

```
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse = (3 * denominator) ^ 2 (contracts/libraries/LibPRBMath.sol#246)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#250)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#251)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#252)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#253)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#254)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#255)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -prod0 = prod0 / twos (contracts/libraries/LibPRBMath.sol#234)
        -result = prod0 * inverse (contracts/libraries/LibPRBMath.sol#261)
LibUnripeSilo._removeUnripeLPDeposit(address,uint32,uint256) (contracts/libraries/Silo/LibUnripeSilo.sol#212-289) performs a multiplication on the result of a division:
        -removed = amount.mul(bdv1).div(amount1) (contracts/libraries/Silo/LibUnripeSilo.sol#230)
        -s.a[account].lp.depositSeeds[season] = s.a[account].lp.depositSeeds[season].sub(removed.mul(4)) (contracts/libraries/Silo/LibUnripeSilo.sol#231-235)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

Contract locking ether found:
        Contract WhitelistFacet (contracts/beanstalk/silo/WhitelistFacet.sol#16-62) has payable functions:
         - WhitelistFacet.dewhitelistToken(address) (contracts/beanstalk/silo/WhitelistFacet.sol#32-35)
         - WhitelistFacet.whitelistToken(address,bytes4,uint32,uint32) (contracts/beanstalk/silo/WhitelistFacet.sol#37-50)
         - WhitelistFacet.updateStalkPerBdvPerSeasonForToken(address,uint32) (contracts/beanstalk/silo/WhitelistFacet.sol#52-61)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether

LibDiamond.addFunctions(address,bytes4[]).selectorIndex (contracts/libraries/LibDiamond.sol#129) is a local variable never initialized
LibDiamond.diamondCut(IDiamondCut.FacetCut[],address,bytes).facetIndex (contracts/libraries/LibDiamond.sol#104) is a local variable never initialized
LibDiamond.removeFunctions(address,bytes4[]).selectorIndex (contracts/libraries/LibDiamond.sol#162) is a local variable never initialized
LibLegacyTokenSilo._claimWithdrawals(address,address,uint32[]).i (contracts/libraries/Silo/LibLegacyTokenSilo.sol#587) is a local variable never initialized
LibDiamond.replaceFunctions(address,bytes4[]).selectorIndex (contracts/libraries/LibDiamond.sol#147) is a local variable never initialized
LibLegacyTokenSilo._mowAndMigrate(address,address[],uint32[][],uint256[][]).perTokenData (contracts/libraries/Silo/LibLegacyTokenSilo.sol#371) is a local variable never initialized
LibSilo._removeDepositsFromAccount(address,address,int96[],uint256[]).i (contracts/libraries/Silo/LibSilo.sol#555) is a local variable never initialized
LibLegacyTokenSilo._mowAndMigrate(address,address[],uint32[][],uint256[][]).migrateData (contracts/libraries/Silo/LibLegacyTokenSilo.sol#367) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

LibUnripeSilo.unripeLPDeposit(address,uint32).bdv (contracts/libraries/Silo/LibUnripeSilo.sol#306) is written in both
        (amount,bdv) = getBeanEthUnripeLP(account,season) (contracts/libraries/Silo/LibUnripeSilo.sol#312)
        bdv = uint256(s.a[account].legacyDeposits[C.unripeLPAddress()][season].bdv).add(legBdv) (contracts/libraries/Silo/LibUnripeSilo.sol#329-331)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#write-after-write

Variable 'LibLegacyTokenSilo._mowAndMigrate(address,address[],uint32[][],uint256[][]).perDepositData (contracts/libraries/Silo/LibLegacyTokenSilo.sol#376)' in LibLegacyTokenSilo._mowAndMigrate(address,address[],uin
/libraries/Silo/LibLegacyTokenSilo.sol#345-434) potentially used before declaration: j < seasons[i].length (contracts/libraries/Silo/LibLegacyTokenSilo.sol#375)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in LibTokenSilo.deposit(address,address,int96,uint256) (contracts/libraries/Silo/LibTokenSilo.sol#108-116):
        External calls:
        - bdv = beanDenominatedValue(token,amount) (contracts/libraries/Silo/LibTokenSilo.sol#114)
                - (success,data) = address(this).call(callData) (contracts/libraries/Silo/LibTokenSilo.sol#286-288)
        Event emitted after the call(s):
        - AddDeposit(account,token,stem,amount,bdv) (contracts/libraries/Silo/LibTokenSilo.sol#283)
                - depositWithBDV(account,token,stem,amount,bdv) (contracts/libraries/Silo/LibTokenSilo.sol#115)
        - TransferSingle(msg.sender,address(0),account,uint256(depositId),amount) (contracts/libraries/Silo/LibTokenSilo.sol#195-201)
                - depositWithBDV(account,token,stem,amount,bdv) (contracts/libraries/Silo/LibTokenSilo.sol#115)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

LibAppStorage.diamondStorage() (contracts/libraries/LibAppStorage.sol#15-19) uses assembly
        - INLINE ASM (contracts/libraries/LibAppStorage.sol#16-18)
LibBytes.toUint8(bytes,uint256) (contracts/libraries/LibBytes.sol#18-28) uses assembly
        - INLINE ASM (contracts/libraries/LibBytes.sol#23-25)
LibBytes.toUint32(bytes,uint256) (contracts/libraries/LibBytes.sol#34-44) uses assembly
        - INLINE ASM (contracts/libraries/LibBytes.sol#39-41)
```

SiloFacet.sol

```
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse = (3 * denominator) ^ 2 (contracts/libraries/LibPRBMath.sol#246)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#250)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#251)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#252)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#253)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#254)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -denominator = denominator / twos (contracts/libraries/LibPRBMath.sol#231)
        -inverse *= 2 - denominator * inverse (contracts/libraries/LibPRBMath.sol#255)
LibPRBMath.mulDiv(uint256,uint256,uint256) (contracts/libraries/LibPRBMath.sol#185-263) performs a multiplication on the result of a division:
        -prod0 = prod0 / twos (contracts/libraries/LibPRBMath.sol#234)
        -result = prod0 * inverse (contracts/libraries/LibPRBMath.sol#261)
LibUnripeSilo._removeUnripeLPDeposit(address,uint32,uint256) (contracts/libraries/Silo/LibUnripeSilo.sol#212-289) performs a multiplication on the result of a division:
        -removed = amount.mul(bdv1).div(amount1) (contracts/libraries/Silo/LibUnripeSilo.sol#238)
        -s.a[account].lp.depositSeeds[season] = s.a[account].lp.depositSeeds[season].sub(removed.mul(4)) (contracts/libraries/Silo/LibUnripeSilo.sol#231-235)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

LibTransfer.sendToken(IERC20,uint256,address,LibTransfer.To) (contracts/libraries/Token/LibTransfer.sol#73-83) uses a dangerous strict equality:
        - amount == 0 (contracts/libraries/Token/LibTransfer.sol#79)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in Silo._claimPlenty(address) (contracts/beanstalk/silo/SiloFacet/Silo.sol#147-154):
        External calls:
        - C.threeCrv().safeTransfer(account,plenty) (contracts/beanstalk/silo/SiloFacet/Silo.sol#150)
        State variables written after the call(s):
        - delete s.a[account].sop.plenty (contracts/beanstalk/silo/SiloFacet/Silo.sol#151)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

SiloFacet.mowMultiple(address,address[]).i (contracts/beanstalk/silo/SiloFacet/SiloFacet.sol#270) is a local variable never initialized
LibSilo._removeDepositsFromAccount(address,address,int96[],uint256[]).i (contracts/libraries/Silo/LibSilo.sol#555) is a local variable never initialized
LibLegacyTokenSilo._claimWithdrawals(address,address,uint32[]).i (contracts/libraries/Silo/LibLegacyTokenSilo.sol#587) is a local variable never initialized
LibLegacyTokenSilo._mowAndMigrate(address,address[],uint32[][],uint256[][]).perTokenData (contracts/libraries/Silo/LibLegacyTokenSilo.sol#371) is a local variable never initialized
SiloFacet.safeBatchTransferFrom(address,address,uint256[],uint256[],bytes).i (contracts/beanstalk/silo/SiloFacet/SiloFacet.sol#243) is a local variable never initialized
TokenSilo._transferDeposits(address,address,address,int96[],uint256[]).i (contracts/beanstalk/silo/SiloFacet/TokenSilo.sol#335) is a local variable never initialized
LibLegacyTokenSilo._mowAndMigrate(address,address[],uint32[][],uint256[][]).migrateData (contracts/libraries/Silo/LibLegacyTokenSilo.sol#367) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

LibUnripeSilo.unripeLPDeposit(address,uint32).bdv (contracts/libraries/Silo/LibUnripeSilo.sol#306) is written in both
        [amount,bdv] = getBeanEthUnripeLP(account,season) (contracts/libraries/Silo/LibUnripeSilo.sol#312)
        bdv = uint256(s.a[account].legacyDeposits[C.unripeLPAddress()][season].bdv).add(legBdv) (contracts/libraries/Silo/LibUnripeSilo.sol#329-331)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#write-after-write

Variable 'LibLegacyTokenSilo._mowAndMigrate(address,address[],uint32[][],uint256[][]).perDepositData (contracts/libraries/Silo/LibLegacyTokenSilo.sol#376)' in LibLegacyTokenSilo._mowAndMigrate(address,address[],uin
/libraries/Silo/LibLegacyTokenSilo.sol#345-434) potentially used before declaration: j < seasons[i].length (contracts/libraries/Silo/LibLegacyTokenSilo.sol#375)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in Silo._claimPlenty(address) (contracts/beanstalk/silo/SiloFacet/Silo.sol#147-154):
        External calls:
        - C.threeCrv().safeTransfer(account,plenty) (contracts/beanstalk/silo/SiloFacet/Silo.sol#150)
        Event emitted after the call(s):
        - ClaimPlenty(account,plenty) (contracts/beanstalk/silo/SiloFacet/Silo.sol#153)
Reentrancy in SiloFacet.deposit(address,uint256,LibTransfer.From) (contracts/beanstalk/silo/SiloFacet/SiloFacet.sol#50-68):
        External calls:
        - amount = LibTransfer.receiveToken(IERC20(token),_amount,msg.sender,mode) (contracts/beanstalk/silo/SiloFacet/SiloFacet.sol#61-66)
        - (bdv,stem) = _deposit(msg.sender,token,amount) (contracts/beanstalk/silo/SiloFacet/SiloFacet.sol#67)
                - stalk = LibTokenSilo.deposit(account,token,stem = LibTokenSilo.stemTipForToken(token),amount) (contracts/beanstalk/silo/SiloFacet/TokenSilo.sol#162-167)
                - (success,data) = address(this).call(callData) (contracts/libraries/Silo/LibTokenSilo.sol#286-288)
        Event emitted after the call(s):
        - AddDeposit(account,token,stem,amount,bdv) (contracts/libraries/Silo/LibTokenSilo.sol#203)
                - (bdv,stem) = _deposit(msg.sender,token,amount) (contracts/beanstalk/silo/SiloFacet/SiloFacet.sol#67)
        - StalkBalanceChanged(account,int256(stalk),int256(roots)) (contracts/libraries/Silo/LibSilo.sol#164)
```

AppStorage.sol

```
Different versions of Solidity are used:
        - Version used: ['=0.7.6', '>=0.6.0<0.8.0']
        - >=0.6.0<0.8.0 (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#3)
        - >=0.6.0<0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#3)
        - >=0.6.0<0.8.0 (lib/openzeppelin-contracts/contracts/utils/Counters.sol#3)
        - =0.7.6 (contracts/beanstalk/AppStorage.sol#3)
        - ABIEncoderV2 (contracts/beanstalk/AppStorage.sol#4)
        - ABIEncoderV2 (contracts/interfaces/IDiamondCut.sol#2)
        - =0.7.6 (contracts/interfaces/IDiamondCut.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Counters.current(Counters.Counter) (lib/openzeppelin-contracts/contracts/utils/Counters.sol#28-30) is never used and should be removed
Counters.decrement(Counters.Counter) (lib/openzeppelin-contracts/contracts/utils/Counters.sol#37-39) is never used and should be removed
Counters.increment(Counters.Counter) (lib/openzeppelin-contracts/contracts/utils/Counters.sol#32-35) is never used and should be removed
SafeMath.add(uint256,uint256) (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#85-89) is never used and should be removed
SafeMath.div(uint256,uint256) (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#135-138) is never used and should be removed
SafeMath.div(uint256,uint256,string) (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#190-193) is never used and should be removed
SafeMath.mod(uint256,uint256) (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#152-155) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#210-213) is never used and should be removed
SafeMath.mul(uint256,uint256) (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#116-121) is never used and should be removed
SafeMath.sub(uint256,uint256) (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#101-104) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#170-173) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#24-28) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#68-73) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#70-73) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#45-63) is never used and should be removed
SafeMath.trySub(uint256,uint256) (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#35-38) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version>=0.6.0<0.8.0 (lib/openzeppelin-contracts/contracts/math/SafeMath.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (lib/openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#3) is too complex
Pragma version>=0.6.0<0.8.0 (lib/openzeppelin-contracts/contracts/utils/Counters.sol#3) is too complex
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

- As a result of the tests carried out with the Slither tool, some results were obtained and reviewed by Halborn. Based on the results reviewed, the majority of vulnerabilities were determined to be false positives.

MythX Results:

Report for contracts/beanstalk/silo/SiloFacet/SiloExit.sol
https://dashboard.mythx.io/#/console/analyses/469834e9-5131-41db-bf06-d3f44374ed71
https://dashboard.mythx.io/#/console/analyses/3f667a62-6511-4d6b-b23e-a4689bedefe0
https://dashboard.mythx.io/#/console/analyses/ad57b6b0-5b2f-4254-a1e7-48b840b3d104
https://dashboard.mythx.io/#/console/analyses/f9b884bb-af59-440c-92ec-512ea3822cd1

Line	SWC Title	Severity	Short Description
171	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randomness.

Report for contracts/beanstalk/silo/SiloFacet/SiloFacet.sol
https://dashboard.mythx.io/#/console/analyses/469834e9-5131-41db-bf06-d3f44374ed71

Line	SWC Title	Severity	Short Description
5	(SWC-103) Floating Pragma	Low	A floating pragma is set.
28	(SWC-123) Requirement Violation	Low	Requirement violation.

Report for contracts/libraries/Silo/LibSilo.sol
https://dashboard.mythx.io/#/console/analyses/f9b884bb-af59-440c-92ec-512ea3822cd1
https://dashboard.mythx.io/#/console/analyses/3a2d70c3-0bef-44c9-9b0d-20c968ababf9
https://dashboard.mythx.io/#/console/analyses/469834e9-5131-41db-bf06-d3f44374ed71
https://dashboard.mythx.io/#/console/analyses/ad57b6b0-5b2f-4254-a1e7-48b840b3d104
https://dashboard.mythx.io/#/console/analyses/3f667a62-6511-4d6b-b23e-a4689bedefe0

Line	SWC Title	Severity	Short Description
237	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randomness.
612	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randomness.

Report for contracts/libraries/Silo/LibTokenSilo.sol
https://dashboard.mythx.io/#/console/analyses/469834e9-5131-41db-bf06-d3f44374ed71

Line	SWC Title	Severity	Short Description
286	(SWC-123) Requirement Violation	Low	Requirement violation.

Report for contracts/libraries/LibInternal.sol
https://dashboard.mythx.io/#/console/analyses/ad57b6b0-5b2f-4254-a1e7-48b840b3d104
https://dashboard.mythx.io/#/console/analyses/3f667a62-6511-4d6b-b23e-a4689bedefe0
https://dashboard.mythx.io/#/console/analyses/f9b884bb-af59-440c-92ec-512ea3822cd1

Line	SWC Title	Severity	Short Description
20	(SWC-109) Uninitialized Storage Pointer	Medium	Dangerous use of uninitialized storage variables.

Report for contracts/beanstalk/silo/WhitelistFacet.sol
https://dashboard.mythx.io/#/console/analyses/3a2d70c3-0bef-44c9-9b0d-20c968ababf9

Line	SWC Title	Severity	Short Description
5	(SWC-103) Floating Pragma	Low	A floating pragma is set.

- No major issues found by the MythX tool.

THANK YOU FOR CHOOSING

// HALBORN