Beanstalk - BIP24

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **August 22nd, 2022 - September 9th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/22/2022	Roberto Reigada
0.2	Document Updates	09/08/2022	Roberto Reigada
0.3	Draft Review	09/08/2022	Gabi Urrutia
1.0	Remediation Plan	09/15/2022	Roberto Reigada
1.1	Remediation Plan Review	09/16/2022	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
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Beanstalk engaged Halborn to conduct a security audit on their smart contracts beginning on August 22nd, 2022 and ending on September 9th, 2022. The security assessment was scoped to the smart contracts provided in the GitHub repository BeanstalkFarms/Beanstalk/tree/bip-24.

1.2 AUDIT SUMMARY

The team at Halborn was provided 2 weeks for the engagement and assigned a full-time security engineer to audit 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 audit is to:

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

In summary, Halborn identified some security risks that were successfully addressed by the Beanstalk team.

1.3 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 audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.

3 - May cause a partial impact or loss to many.

2 - May cause temporary impact or loss.

1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL

9 - 8 - HIGH

7 - 6 - MEDIUM

5 - 4 - LOW

3 - 1 - VERY LOW AND INFORMATIONAL

EXECUTIVE OVERVIEW

1.4 SCOPE

IN-SCOPE:
The security assessment was scoped to the code changes performed in these smart contracts since our last audit Commit ID:

- MarketplaceFacet.sol
- SeasonFacet.sol
- SiloFacet.sol
- WhitelistFacet.sol
- UnripeFacet.sol
- TokenFacet.sol
- PauseFacet.sol
- OwnershipFacet.sol
- FieldFacet.sol
- FertilizerFacet.sol
- FarmFacet.sol
- DiamondLoupeFacet.sol
- DiamondCutFacet.sol
- CurveFacet.sol
- ConvertFacet.sol
- BDVFacet.sol
- FundraiserFacet.sol
- AppStorage.sol
- Diamond.sol
- BeanstalkPrice.sol
- CurvePrice.sol
- P.sol

Initial commit ID:
- f3dcb644604b117735dc3917bc3c9d5e8749f476

These were all the code changes done between 1447fa2c0d42c73345a38edb4f4dad076392f429 and f3dcb644604b117735dc3917bc3c9d5e8749f476:

Listing.sol, MarketplaceFacet.sol, Order.sol
- Minor change in _cancelPodListing() function. This function now accepts an address to cancel the listing of that address.

Oracle.sol
- A new event was added:
event MetapoolOracle(uint32 indexed season, int256 deltaB, uint256[2] balances);

SeasonFacet.sol
- Minor change related to Sunrise event. It is now emitted in the stepSeason() function instead of sunrise() although as stepSeason() is called within the sunrise() function, there is no difference.

Sun.sol
- Minor changes to 2 events: event Rewards & event Soil.

Silo.sol
- Renamed _earn() function to _plant() function.
- Renamed earn event to plant event.
- Minor change in the _earn() function.

SiloFacet.sol
- A new Silo deposit approval system was implemented.

TokenSilo.sol
- New event DepositApproval was added.
- The functions _spendDepositAllowance(), _approveDeposit() and depositAllowance() were added.

ConvertFacet.sol
- Added a return value to the convert() function.

CurveFacet.sol
- Added Curve3Pool checks.

FertilizerFacet.sol
- Added some view functions: getCurrentHumidity() and getFertilizers().

FieldFacet.sol

- Minor change in the _sow() function.

OwnershipFacet.sol

- Added a 2-step process for the ownership transfer.

UnripeFacet.sol

- Renamed Ripen to Chop.
- Renamed ClaimUnripe to Pick.
- Added 2 view functions: picked() and getUnderlyingToken().
- Major changes in the pick() function. fromMode was also added as a parameter to this function.

Internalizer.sol

- Minor changes in the setURI() function.

LibConvert.sol

- Lambda if case added.

LibConvertData.sol

- Added LAMBDA_LAMBDA case to the ConvertKind enum.
- Added lambdaConvert() pure function.

LibCurveConvert.sol

- Minor change in lpToPeg() function.

LibLambdaConvert.sol

- Library implemented from scratch.

LibMetaCurveConvert.sol

- Added lpToPeg(), calcLPTokenAmount() and toPegWithFee() functions.

LibBeanMetaCurve.sol

- Added getXP0() function.

LibCurve.sol

- Added getYD() function.

EXECUTIVE OVERVIEW

LibCurveOracle.sol

- Added a new event: MetapoolOracle.
- Updated mintPrecision from 240 to 100.

LibSilo.sol

- Minor change in decrementBalanceOfStalk() function.

LibTokenSilo.sol

- Minor change in beanDenominatedValue() function.

LibWhitelist.sol

- 2 constants were removed: BEAN_LUSD_STALK and BEAN_LUSD_SEEDS.

LibApprove.sol

- Updated the approveToken() function.

LibTransfer.sol

- Added a new function: burnToken().

LibFertilizer.sol

- Minor changes to getHumidity(), addUnderlying() and remainingRecapitalization() functions.

BeanstalkPrice.sol, CurvePrice.sol, P.sol

- Implemented from scratch.

Final commit ID:

- 59162c1344492c5160005ecd3eafb062c3d45668

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	0	1	1	2

LIKELIHOOD

IMPACT

(HAL-02)				(HAL-01)
(HAL-03)				
(HAL-04) (HAL-05)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - UNDERLYING TOKENS CAN BE DRAINED THROUGH THE UNRIPEFACET.CHOP FUNCTION	Critical	SOLVED - 09/16/2022
HAL02 - ROOTS CAN BE DRAINED THROUGH THE SILOFACET.TRANSFERDEPOSITS FUNCTION	Medium	SOLVED - 09/16/2022
HAL03 - OVERFLOW IN INCREASEDEPOSITALLOWANCE FUNCTION	Low	SOLVED - 09/16/2022
HAL04 - SILOFACET.CLAIMPLENTY FUNCTION ALLOWS ANYONE TO CLAIM ON YOUR BEHALF	Informational	SOLVED - 09/16/2022
HAL05 - APPROVETOKEN FUNCTION ACTS AS A SAFEINCREASEALLOWANCE CALL	Informational	SOLVED - 09/16/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) UNDERLYING TOKENS CAN BE DRAINED THROUGH THE UNRIPEFACET.CHOP FUNCTION - CRITICAL

Description:

In the UnripeFacet, the chop() function is used to burn unripeTokens in order to receive in exchange an underlyingToken like, for example, Beans:

Listing 1: UnripeFacet.sol (Line 61)

```
51 function chop(
52     address unripeToken,
53     uint256 amount,
54     LibTransfer.From fromMode,
55     LibTransfer.To toMode
56 ) external payable nonReentrant returns (uint256 underlyingAmount) {
57     underlyingAmount = getPenalizedUnderlying(unripeToken, amount);
58
59     LibUnripe.decrementUnderlying(unripeToken, underlyingAmount);
60
61     LibTransfer.burnToken(IBean(unripeToken), amount, msg.sender, fromMode);
62
63     address underlyingToken = s.u[unripeToken].underlyingToken;
64
65     IERC20(underlyingToken).sendToken(underlyingAmount, msg.sender, toMode);
66
67     emit Chop(msg.sender, unripeToken, amount, underlyingAmount);
68 }
```

The burn of the unripeTokens is done through the LibTransfer.burnToken() call:

Listing 2: LibTransfer.sol (Lines 87,95)

```
82 function burnToken(
83     IBean token,
84     uint256 amount,
85     address sender,
86     From mode
87 ) internal returns (uint256 burnt) {
88     // burnToken only can be called with Unripe Bean, Unripe Bean:3Crv or Bean token, which are all Beanstalk tokens.
89     // Beanstalk's ERC-20 implementation uses OpenZeppelin's ERC20Burnable
90     // which reverts if burnFrom function call cannot burn full amount.
91     if (mode == From.EXTERNAL) {
92         token.burnFrom(sender, amount);
93         burnt = amount;
94     } else {
95         burnt = LibTransfer.receiveToken(token, amount, sender, mode);
96         token.burn(burnt);
97     }
98 }
```

As we can see, the `LibTransfer.burnToken()` function returns the actual amount of tokens that were burnt.

The `LibTransfer.From fromMode` has 4 different modes:

- EXTERNAL
- INTERNAL
- EXTERNAL_INTERNAL
- INTERNAL_TOLERANT

With the `INTERNAL_TOLERANT` fromMode tokens will be collected from the user's Internal Balance and the transaction will not fail if there is not enough tokens there.

This `INTERNAL_TOLERANT` fromMode can be used in the `UnripeFacet.chop()` call. As the `chop()` function is not checking the return value of the

LibTransfer.burnToken() the contract will always assume that the full amount is being burnt when that will not always be true. If a user actually has 0 unripeTokens and uses the INTERNAL_TOLERANT fromMode, no tokens will be burned at all but the full amount of underlyingTokens will be sent to the user.

Proof of Concept:

This test was done forking the Ethereum mainnet on block 15465331 (Sep-03-2022 12:16:18 PM +UTC):

```
Calling -> contract_UnripeFacet = Contract.from_abi('UnripeFacet', '0xc1e088fc1323b20bcbee9bd1b9fc9546db5624c5', UnripeFacet.abi, owner=owner)
contract_BEAN.balanceOf(user1) -> 0
contract_UNRIPEBEAN.balanceOf(user1) -> 0

Calling -> contract_UnripeFacet.chop(contract_UNRIPEBEAN.address, 1000000000_000000, 3, 0, {'from': user1, 'value': 0})
Transaction sent: 0x48d539fa2b2e2db261eafd8587f509a30ffc6635e4df2b90d371d4547b8209c4
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 0
  UnripeFacet.chop confirmed   Block: 15465345   Gas used: 88776 (0.01%)

contract_BEAN.balanceOf(user1) -> 4880867830117
contract_UNRIPEBEAN.balanceOf(user1) -> 0


Calling -> contract_UnripeFacet.chop(contract_UNRIPEBEAN.address, 1000000000_000000, 3, 0, {'from': user1, 'value': 0})
Transaction sent: 0x4182547af19b6c829c749b2e6e692c481ad685459b99aead5edde9934c996423
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 1
  UnripeFacet.chop confirmed   Block: 15465346   Gas used: 62976 (0.01%)

contract_BEAN.balanceOf(user1) -> 8047676452716
contract_UNRIPEBEAN.balanceOf(user1) -> 0
```

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

It is recommended to save the return value of the LibTransfer.burnToken() call and overwrite the amount variable with that return as shown below:

Listing 3: UnripeFacet.sol (Line 57)

```
function chop(
    address unripeToken,
    uint256 amount,
    LibTransfer.From fromMode,
    LibTransfer.To toMode
) external payable nonReentrant returns (uint256 underlyingAmount)
↳ {
```

```
    amount = LibTransfer.burnToken(IBean(unripeToken), amount, msg.sender, fromMode);

    underlyingAmount = getPenalizedUnderlying(unripeToken, amount);

    LibUnripe.decrementUnderlying(unripeToken, underlyingAmount);

    address underlyingToken = s.u[unripeToken].underlyingToken;

    IERC20(underlyingToken).sendToken(underlyingAmount, msg.sender, toMode);

    emit Chop(msg.sender, unripeToken, amount, underlyingAmount);
}
```

Remediation Plan:

SOLVED: The Beanstalk team fixed the issue by now taking also considering the return value of the LibTransfer.burnToken().

3.2 (HAL-02) ROOTS CAN BE DRAINED THROUGH THE SILOFACET.TRANSFERDEPOSITS FUNCTION - MEDIUM

Description:

In the SiloFacet, the transferDeposits() function is used to transfer multiple deposits to a new wallet:

Listing 4: SiloFacet.sol (Lines 108-110)

```
100 function transferDeposits(
101     address sender,
102     address recipient,
103     address token,
104     uint32[] calldata seasons,
105     uint256[] calldata amounts
106 ) external payable nonReentrant {
107     if (sender != msg.sender) {
108         for (uint256 i = 0; i < amounts.length; i++) {
109             _spendDepositAllowance(sender, msg.sender, token,
    ↳ amounts[i]);
110         }
111     }
112     _update(sender);
113     // Need to update the recipient's Silo as well.
114     _update(recipient);
115     _transferDeposits(sender, recipient, token, seasons, amounts);
116 }
```

Although if the seasons and amounts array are both empty or if the amounts array contains zeros the _spendDepositAllowance() function would be skipped and the _transferDeposits() line would be executed:

Listing 5: TokenSilo.sol (Lines 359-364)

```
319 function _transferDeposits(
320     address sender,
321     address recipient,
322     address token,
323     uint32[] calldata seasons,
324     uint256[] calldata amounts
325 ) internal {
326     require(
327         seasons.length == amounts.length,
328         "Silo: Crates, amounts are diff lengths."
329     );
330     AssetsRemoved memory ar;
331     for (uint256 i; i < seasons.length; ++i) {
332         uint256 crateBdv = LibTokenSilo.removeDeposit(
333             sender,
334             token,
335             seasons[i],
336             amounts[i]
337         );
338         LibTokenSilo.addDeposit(
339             recipient,
340             token,
341             seasons[i],
342             amounts[i],
343             crateBdv
344         );
345         ar.bdvRemoved = ar.bdvRemoved.add(crateBdv);
346         ar.tokensRemoved = ar.tokensRemoved.add(amounts[i]);
347         ar.stalkRemoved = ar.stalkRemoved.add(
348             LibSilo.stalkReward(
349                 crateBdv.mul(s.ss[token].seeds),
350                 _season() - seasons[i]
351             )
352         );
353     }
354     ar.seedsRemoved = ar.bdvRemoved.mul(s.ss[token].seeds);
355     ar.stalkRemoved = ar.stalkRemoved.add(
356         ar.bdvRemoved.mul(s.ss[token].stalk)
357     );
358     emit RemoveDeposits(sender, token, seasons, amounts, ar.tokensRemoved);
359     LibSilo.transferSiloAssets(
360         sender,
```

```
            recipient,
            ar.seedsRemoved,
            ar.stalkRemoved
        );
}
```

This function would call the `LibSilo.transferSiloAssets()` function:

Listing 6: LibSilo.sol (Line 58)

```
52 function transferSiloAssets(
53     address sender,
54     address recipient,
55     uint256 seeds,
56     uint256 stalk
57 ) internal {
58     transferStalk(sender, recipient, stalk);
59     transferSeeds(sender, recipient, seeds);
60 }
```

The `transferStalk()` function would be executed with the parameter `stalk = 0`:

Listing 7: LibSilo.sol (Line 129)

```
121     function transferStalk(
122         address sender,
123         address recipient,
124         uint256 stalk
125     ) private {
126         AppStorage storage s = LibAppStorage.diamondStorage();
127         uint256 roots = stalk == s.a[sender].s.stalk
128             ? s.a[sender].roots
129             : s.s.roots.sub(1).mul(stalk).div(s.s.stalk).add(1);
130
131         s.a[sender].s.stalk = s.a[sender].s.stalk.sub(stalk);
132         s.a[sender].roots = s.a[sender].roots.sub(roots);
133
134         s.a[recipient].s.stalk = s.a[recipient].s.stalk.add(stalk)
    ↳ ;
135         s.a[recipient].roots = s.a[recipient].roots.add(roots);
```

```
136     }
```

With `stalk` parameter being 0, roots would be equal to 1. This means that after every call, 1 stalk would be decreased from the sender and added to the recipient.

Technically, any user would be able to drain the roots of any other user, although, the gas costs are much too high for this attack to be worth as the attacker would only be able to steal 1 root per call.

Proof of Concept:

```
contract_SiloFacet.depositAllowance(user1, user2, contract_BEAN) -> 0
contract_SiloFacet.balanceOfEarnedBeans(user1) -> 0
contract_SiloFacet.balanceOfEarnedBeans(user2) -> 0
contract_SiloFacet.balanceOfEarnedSeeds(user1) -> 0
contract_SiloFacet.balanceOfEarnedSeeds(user2) -> 0
contract_SiloFacet.balanceOfEarnedStalk(user1) -> 0
contract_SiloFacet.balanceOfEarnedStalk(user2) -> 0
contract_SiloFacet.balanceOfGrownStalk(user1) -> 0
contract_SiloFacet.balanceOfGrownStalk(user2) -> 0
contract_SiloFacet.balanceOfPlenty(user1) -> 0
contract_SiloFacet.balanceOfPlenty(user2) -> 0
contract_SiloFacet.balanceOfRainRoots(user1) -> 0
contract_SiloFacet.balanceOfRainRoots(user2) -> 0
contract_SiloFacet.balanceOfRoots(user1) -> 1000000000000000000000000000
contract_SiloFacet.balanceOfRoots(user2) -> 0
contract_SiloFacet.balanceOfSeeds(user1) -> 1000000000000000
contract_SiloFacet.balanceOfSeeds(user2) -> 0
contract_SiloFacet.balanceOfSop(user1) -> (0, 0, 0, 0, 0)
contract_SiloFacet.balanceOfSop(user2) -> (0, 0, 0, 0, 0)
contract_SiloFacet.balanceOfStalk(user1) -> 1000000000000000
contract_SiloFacet.balanceOfStalk(user2) -> 0

Calling -> t1 = contract_SiloFacet.transferDeposits(user1.address, user2.address, contract_BEAN.address, [], [], {'from': user2, 'value': 0})
Transaction sent: 0xe8a9d60dc6624766c45ec518fdf7ba27aa8716acddd7611da8330c9bfedcc7e8
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 0
  Transaction confirmed   Block: 15484547   Gas used: 81659 (0.01%)

contract_SiloFacet.balanceOfEarnedBeans(user1) -> 0
contract_SiloFacet.balanceOfEarnedBeans(user2) -> 0
contract_SiloFacet.balanceOfEarnedSeeds(user1) -> 0
contract_SiloFacet.balanceOfEarnedSeeds(user2) -> 0
contract_SiloFacet.balanceOfEarnedStalk(user1) -> 0
contract_SiloFacet.balanceOfEarnedStalk(user2) -> 0
contract_SiloFacet.balanceOfGrownStalk(user1) -> 0
contract_SiloFacet.balanceOfGrownStalk(user2) -> 0
contract_SiloFacet.balanceOfPlenty(user1) -> 0
contract_SiloFacet.balanceOfPlenty(user2) -> 0
contract_SiloFacet.balanceOfRainRoots(user1) -> 0
contract_SiloFacet.balanceOfRainRoots(user2) -> 0
contract_SiloFacet.balanceOfRoots(user1) -> 999999999999999999999999999
contract_SiloFacet.balanceOfRoots(user2) -> 1
contract_SiloFacet.balanceOfSeeds(user1) -> 1000000000000000
contract_SiloFacet.balanceOfSeeds(user2) -> 0
contract_SiloFacet.balanceOfSop(user1) -> (0, 0, 0, 0, 0)
contract_SiloFacet.balanceOfSop(user2) -> (0, 0, 0, 0, 0)
contract_SiloFacet.balanceOfStalk(user1) -> 1000000000000000
contract_SiloFacet.balanceOfStalk(user2) -> 0
Calling -> t1 = contract_SiloFacet.transferDeposits(user1.address, user2.address, contract_BEAN.address, [2,2], [0,0], {'from': user2, 'value': 0})
Transaction sent: 0x7a3673a2a84678a37c08e9ec9d7089935bf60770a3092ba2a5755ccf20e19071
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 1
  Transaction confirmed   Block: 15484548   Gas used: 55017 (0.01%)

contract_SiloFacet.balanceOfEarnedBeans(user1) -> 0
contract_SiloFacet.balanceOfEarnedBeans(user2) -> 0
contract_SiloFacet.balanceOfEarnedSeeds(user1) -> 0
contract_SiloFacet.balanceOfEarnedSeeds(user2) -> 0
contract_SiloFacet.balanceOfEarnedStalk(user1) -> 0
contract_SiloFacet.balanceOfEarnedStalk(user2) -> 0
contract_SiloFacet.balanceOfGrownStalk(user1) -> 0
contract_SiloFacet.balanceOfGrownStalk(user2) -> 0
contract_SiloFacet.balanceOfPlenty(user1) -> 0
contract_SiloFacet.balanceOfPlenty(user2) -> 0
contract_SiloFacet.balanceOfRainRoots(user1) -> 0
contract_SiloFacet.balanceOfRainRoots(user2) -> 0
contract_SiloFacet.balanceOfRoots(user1) -> 999999999999999999999999998
contract_SiloFacet.balanceOfRoots(user2) -> 2
contract_SiloFacet.balanceOfSeeds(user1) -> 1000000000000000
contract_SiloFacet.balanceOfSeeds(user2) -> 0
contract_SiloFacet.balanceOfSop(user1) -> (0, 0, 0, 0, 0)
contract_SiloFacet.balanceOfSop(user2) -> (0, 0, 0, 0, 0)
contract_SiloFacet.balanceOfStalk(user1) -> 1000000000000000
contract_SiloFacet.balanceOfStalk(user2) -> 0
```

FINDINGS & TECH DETAILS

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

It is recommended to revert any transferDeposits() call that contains an empty array or a 0 in the amounts array.

Remediation Plan:

SOLVED: The Beanstalk team fixed the issue by adding a require check that enforces arrays to be non-empty.

3.3 (HAL-03) OVERFLOW IN INCREASEDEPOSITALLOWANCE FUNCTION - LOW

Description:

In the SiloFacet, the increaseDepositAllowance() function can overflow:

Listing 8: SiloFacet.sol (Line 61)

```
function increaseDepositAllowance(address spender, address token,
↳ uint256 addedValue) public virtual nonReentrant returns (bool) {
    _approveDeposit(msg.sender, spender, token, depositAllowance(
↳ msg.sender, spender, token) + addedValue);
    return true;
}
```

Let's imagine that the current allowance is already set to for example 1000 tokens and then the same user calls increaseDepositAllowance() and tries to set the maximum uint256 value as the new allowance value:

```
contract_SiloFacet.depositAllowance(user1, user2, contract_BEAN) -> 1000000000

Calling -> contract_SiloFacet.increaseDepositAllowance(user2, contract_BEAN, UINT256_MAX, {'from': user1, 'value': 0})
Transaction sent: 0x526abbda97c6bca1f860ea810e8b824c2c5ed5f9b7af11b26241a528151e5a4b
  Gas price: 0.0 gwei   Gas limit: 600000000   Nonce: 3
  Transaction confirmed   Block: 15484559   Gas used: 35386 (0.01%)

contract_SiloFacet.depositAllowance(user1, user2, contract_BEAN) -> 999999999
```

As we can see, an overflow occurs and the value set as allowance is not the value wanted by the user.

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to use SafeMath in the increaseDepositAllowance() function:

Listing 9: SiloFacet.sol (Line 61)

```
function increaseDepositAllowance(address spender, address token, uint256 addedValue) public virtual nonReentrant returns (bool) {
    _approveDeposit(msg.sender, spender, token, depositAllowance(msg.sender, spender, token).add(addedValue));
    return true;
}
```

Remediation Plan:

SOLVED: The Beanstalk team solved the issue.

3.4 (HAL-04) SILOFACET.CLAIMPLENTY FUNCTION ALLOWS ANYONE TO CLAIM ON YOUR BEHALF - INFORMATIONAL

Description:

In the SiloFacet, the claimPlenty() function allows anyone to claim on behalf of other user:

Listing 10: SiloFacet.sol (Line 157)

```
156 function claimPlenty(address account) external payable {
157     _claimPlenty(account);
158 }
```

Listing 11: Silo.sol (Line 89)

```
86 function _claimPlenty(address account) internal {
87     // Each Plenty is earned in the form of 3Crv.
88     uint256 plenty = s.a[account].sop.plenty;
89     C.threeCrv().safeTransfer(account, plenty);
90     delete s.a[account].sop.plenty;
91
92     emit ClaimPlenty(account, plenty);
93 }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to only allow users to claim for their Plenty. On the other hand, even if the threeCrv() token does not have any hook that opens up for a reentrancy vulnerability, it is recommended to move the deletion of the mapping before the actual safeTransfer() call as shown below:

Listing 12: Silo.sol (Line 89)

```
86 function _claimPlenty(address account) internal {
87     delete s.a[account].sop.plenty;
88     // Each Plenty is earned in the form of 3Crv.
89     uint256 plenty = s.a[account].sop.plenty;
90     C.threeCrv().safeTransfer(account, plenty);
91
92     emit ClaimPlenty(account, plenty);
93 }
```

Remediation Plan:

SOLVED: The Beanstalk team solved the issue.

3.5 (HAL-05) APPROVETOKEN FUNCTION ACTS AS A SAFEINCREASEALLOWANCE CALL - INFORMATIONAL

Description:

In the LibApprove, the approveToken() function instead of setting the allowance to a specific amount, it increases the current allowance by that amount:

Listing 13: LibApprove.sol (Line 25)

```
18 function approveToken(
19     IERC20 token,
20     address spender,
21     uint256 amount
22 ) internal {
23     if (token.allowance(address(this), spender) == type(uint256).
   ↳ max)
24         return;
25     token.safeIncreaseAllowance(spender, amount);
26 }
```

The approveToken() function is called in the functions:
- CurveFacet.exchange()
- CurveFacet.exchangeUnderlying()
- CurveFacet.addLiquidity()

These means that if the allowance is not fully used (hence reset to zero), future approveToken() calls will set the allowance to a value higher than expected.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to consider updating the approveToken() function to use a safeApprove() call instead of safeIncreaseAllowance() as shown below:

Listing 14: LibApprove.sol (Line 25)

```
18 function approveToken(
19     IERC20 token,
20     address spender,
21     uint256 amount
22 ) internal {
23     if (token.allowance(address(this), spender) == type(uint256).
   ↳ max)
24         return;
25     token.safeApprove(spender, 0);
26     token.safeApprove(spender, amount);
27 }
```

Other option is calculating the current allowance and do instead a token.safeIncreaseAllowance(spender, amount - currentAllowance);.

Remediation Plan:

SOLVED: The Beanstalk team solved the issue.

THANK YOU FOR CHOOSING

HALBORN