Root - Token

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **October 6th, 2022 - October 21st, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/21/2022	Pawel Bartunek
0.2	Draft Review	10/21/2022	Kubilay Onur Gungor
0.3	Draft Review	10/21/2022	Gabi Urrutia
1.0	Remediation Plan	10/24/2022	Pawel Bartunek
1.1	Remediation Plan Review	10/24/2022	Kubilay Onur Gungor
1.2	Remediation Plan Review	10/24/2022	Gabi Urrutia
1.3	Document updated - additional commit	10/26/2022	Pawel Bartunek
1.4	Remediation Plan	10/31/2022	Kubilay Onur Gungor
1.5	Remediation Plan Review	10/31/2022	Gabi Urrutia
1.6	Document updated - additional commit	11/09/2022	Pawel Bartunek
1.7	Remediation Plan Review	11/10/2022	Kubilay Onur Gungor
1.8	Remediation Plan Review	11/10/2022	Gabi Urrutia
1.9	Document updated - additional commit	11/15/2022	Pawel Bartunek
2.0	Final Review	11/15/2022	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
Kubilay Onur Gungor	Halborn	Kubilay.Gungor@halborn.com
Pawel Bartunek	Halborn	Pawel.Bartunek@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Root engaged Halborn to conduct a security audit on their smart contracts beginning on October 6th, 2022 and ending on October 21st, 2022. The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided two 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 Root 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 bridge 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 audit:

- Research into architecture and purpose
- Smart contract manual code review and walk-through
- 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
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Local deployment (Hardhat, Remix IDE, Brownie)

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

1.4 SCOPE

IN-SCOPE:
The security assessment was focused on the changes related to Root token implementation, introduced in the following smart contracts:

- protocol/contracts/tokens/Root.sol
- protocol/contracts/libraries/Silo/LibSiloPermit.sol
- protocol/contracts/libraries/Token/LibTokenApprove.sol
- protocol/contracts/libraries/Token/LibTokenPermit.sol
- protocol/contracts/libraries/Token/LibTransfer.sol
- protocol/contracts/farm/AppStorage.sol
- protocol/contracts/farm/facets/SiloFacet/SiloFacet.sol
- protocol/contracts/farm/facets/MarketplaceFacet/MarketplaceFacet.sol
- protocol/contracts/farm/facets/TokenFacet.sol

Branch: root-fdbdv-token
Commit ID: c4bd237a9543c1a130fae18b8ca3fa4ca29d2ffb

Remediation plan:

Branch: root-fdbdv-token
Commit ID: 39cd2f0eff9f7c3e7b731fa3736a5c87a7e74faf

Additional commit to the previous remediation plan, including:

- Renaming of convertLambdaToLambda(s) functions and seed parameter.
- Adding new parameters: minRootsOut to mint and maxRootsIn to redeem functions.
- Adding _transferDeposit helper function.

Commit ID: 9d1fe400326dfdc2018ebef53b354286d1e86b3e

Additional commit to the previous remediation plan, including:

- Change of initial Root token supply to be in 1:1 ratio with Stalk.

Commit ID: 50ca4915010e294cff59590fbf20e7c4ee1edcf8

Additional commit to the previous remediation plan related to EBIP-6, including:

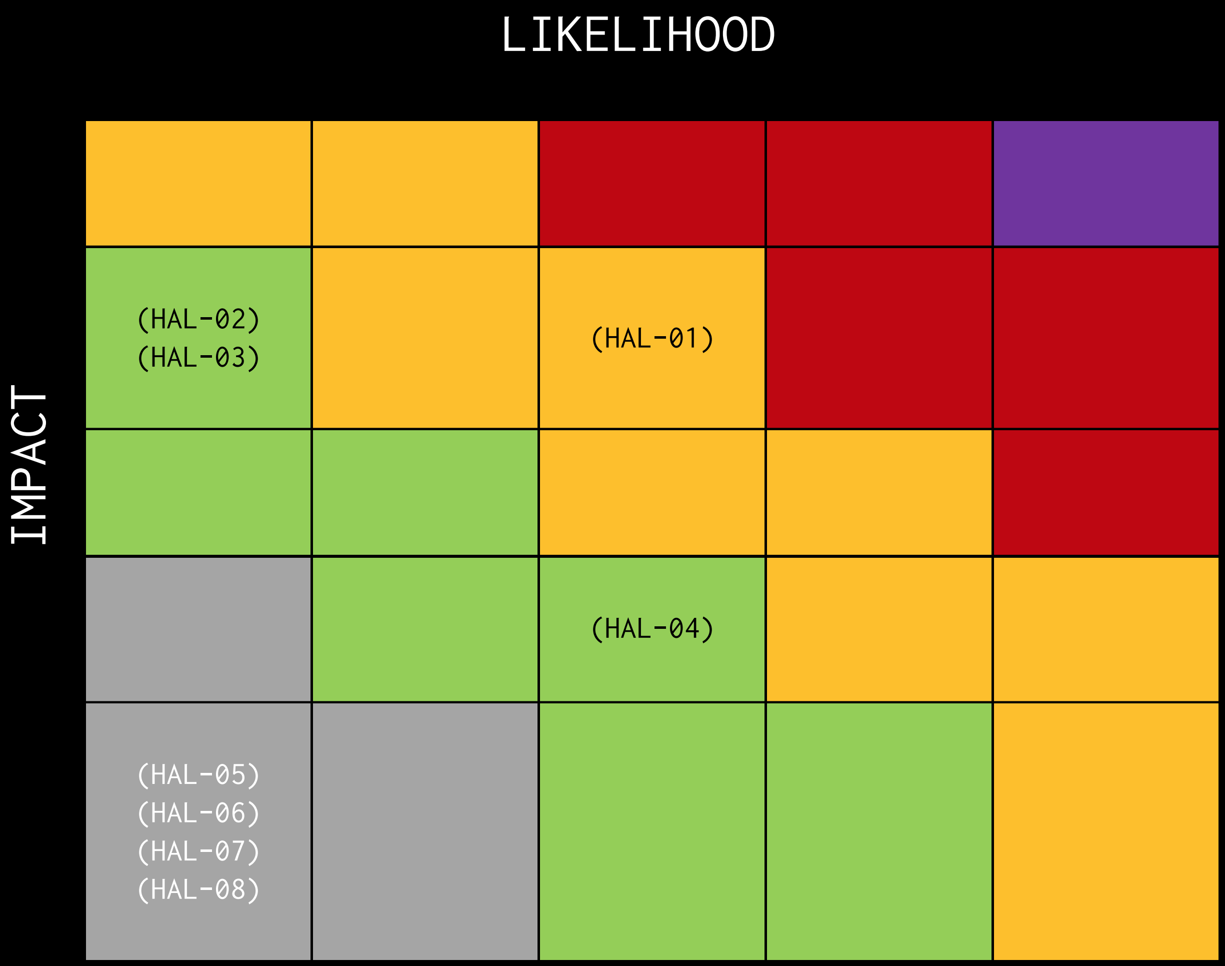
- Change transferTokenFrom reference to transferInternalTokenFrom.
- Restrict transferInternalTokenFrom to INTERNAL mode only.

Commit ID: 808cff496f8161c8d94f7a35e99dbf3ad77142f0

OUT-OF-SCOPE:
Other smart contracts in the repository, external libraries and economical attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	3	4



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - IMPLEMENTATION CONTRACT CAN BE LEFT UNINITIALIZED	Medium	SOLVED - 10/24/2022
HAL-02 - OWNER CAN RENOUNCE OWNERSHIP	Low	SOLVED - 10/24/2022
HAL-03 - LACK OF TWO-STEP TRANSFER OWNERSHIP PATTERN	Low	SOLVED - 10/24/2022
HAL-04 - MISSING ZERO ADDRESS CHECK	Low	SOLVED - 10/24/2022
HAL-05 - SOLC 0.8.4 COMPILER VERSION CONTAINS MULTIPLE BUGS	Informational	SOLVED - 10/24/2022
HAL-06 - MISSING NATSPEC DOCUMENTATION	Informational	SOLVED - 10/24/2022
HAL-07 - FUNCTIONS DO NOT FOLLOW NAMING CONVENTION	Informational	SOLVED - 10/24/2022
HAL-08 - FUNCTIONS COULD BE DEFINED EXTERNAL	Informational	SOLVED - 10/24/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) IMPLEMENTATION CONTRACT CAN BE LEFT UNINITIALIZED - MEDIUM

Description:

The Root contract uses the Initializable module (via UUPSUpgradeable) from OpenZeppelin. During the review, it was discovered that the Root token does not implement a constructor that initializes the implementation contract.

An attacker can take over an uninitialized implementation contract by calling an initialize function directly at the address of the implementation contract.

To prevent leaving an implementation contract uninitialized, the OpenZeppelin's documentation recommends adding the _disableInitializers function in the constructor to lock the contracts when they are automatically deployed:

Listing 1: Initializable.sol

```
/**
 * @dev Locks the contract, preventing any future reinitialization
↳ . This cannot be part of an initializer call.
 * Calling this in the constructor of a contract will prevent that
↳  contract from being initialized or reinitialized
 * to any version. It is recommended to use this to lock
↳ implementation contracts that are designed to be called
 * through proxies.
 */
function _disableInitializers() internal virtual {
    require(!_initializing, "Initializable: contract is
↳ initializing");
    if (_initialized < type(uint8).max) {
        _initialized = type(uint8).max;
        emit Initialized(type(uint8).max);
    }
}
```

Note: The smart contract uses OpenZeppelin Upgradeable version 4.7.3 , which contains a hotfix for the UUPSUpgradeable DoS vulnerability, preventing the upgrade functions from being called directly in the implementation (introduced in 4.3.2). Because the severity of the issue was lowered.

Code Location:

Root.sol

Test case:

Below is an example Hardhat test case, demonstrating another user taking over a deployment contract:

Listing 2: HAL-01 PoC

```
it("HAL-01 Implementation contract not initialized", async
↳ function () {

  const rootImplAddr = await hre.upgrades.erc1967.
↳ getImplementationAddress(this.rootToken.address);
  const rootImpl = await ethers.getContractAt("Root", rootImplAddr
↳ );

  await rootImpl.connect(user).initialize("myroot", "myROOT");

  expect(await rootImpl.owner()).to.be.eq(user.address);
});
```

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

Consider adding the constructor with a call to the _disableInitializers function:

Listing 3: Locking contract with constructor

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
    _disableInitializers();
}
```

Remediation Plan:

SOLVED: The Root team implemented the constructor with a call to the _disableInitializers() function, which locks the implementation contract during deployment.

Commit ID: d4f088a403ba37cfd6aa15cdd02afe3beb579b74

3.2 (HAL-02) OWNER CAN RENOUNCE OWNERSHIP - LOW

Description:

The Owner of the contract is usually the account that deploys the contract. As a result, the Owner can perform some privileged functions. The Root .sol smart contract inherits from the OpenZeppelin OwnableUpgradeable contract, which includes the renounceOwnership function.

Listing 4: contracts/access/OwnableUpgradeable.sol

```
function renounceOwnership() public virtual onlyOwner {
    _transferOwnership(address(0));
}
```

Reference: OwnableUpgradeable.sol

Renouncing ownership before transferring would result in the contract having no Owner, eliminating the ability to call privileged functions.

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is recommended that the Owner cannot call renounceOwnership without transferring the Ownership to another address. In addition, if a multi-signature wallet is used, the call to the renounceOwnership function should be confirmed for two or more users.

As another solution, the Renounce Ownership functionality can be disabled with the following lines of code:

Listing 5: Disabling renounceOwnership function (Line 1)

```
function renounceOwnership () public override onlyOwner {
    revert("can 't renounceOwnership here "); // not possible with
    ↳ this smart contract
}
```

Remediation Plan:

SOLVED: The Root team implemented the renounceOwnership function by disabling the ability to renouncing ownership.

Commit ID: 9015b0f508ef0c8874a309bf8c367cfe5f454def

3.3 (HAL-03) LACK OF TWO-STEP TRANSFER OWNERSHIP PATTERN - LOW

Description:

The current ownership transfer process for the Root contract inherited from OwnableUpgradeable involves the current owner calling the transferOwnership function:

Listing 6: contracts/access/OwnableUpgradeable.sol

```
function transferOwnership(address newOwner) public virtual
↳ onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the
↳ zero address");
    _transferOwnership(newOwner);
}
```

Reference: OwnableUpgradeable.sol

Suppose the nominated account is not valid. In that case, the owner may accidentally transfer ownership to an uncontrolled account, losing access to all functions with the onlyOwner modifier.

Risk Level:

Likelihood - 1
Impact - 4

Recommendation:

It is recommended to implement a two-step process where the owner nominates an account, and the nominated account must call the acceptOwnership() function for the transfer of ownership to succeed. This ensures that the nominated account is valid and active.

Remediation Plan:

SOLVED: The Root team implemented the two-step change ownership pattern.

Commit ID: 57b7e56855c30144026ab5c1f67b3791de032da9

3.4 (HAL-04) MISSING ZERO ADDRESS CHECK - LOW

Description:

Contracts in-scope are missing address validation in constructors and setter functions. It is possible to configure the ZERO address, which may cause issues during execution.

Code Location:

The following functions are not validating, that given address is non-zero:

- Root.sol - addWhitelistToken, token parameter
- Root.sol - setDelegate, _delegateContract, parameter
- Root.sol - _convertLambdaToLambda, token parameter

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Consider adding proper address validation when each state variable assignment is made from user-provided input.

Remediation Plan:

SOLVED: The Root team added validation of address parameters, ensuring that the user cannot set a zero address.

Commit ID: 9bac46b4cbc0af70ff544ef23e1381496043dd01

3.5 (HAL-05) SOLC 0.8.4 COMPILER VERSION CONTAINS MULTIPLE BUGS - INFORMATIONAL

Description:

The scoped contracts have configured the Solidity version to 0.8.4 in Hardhat configuration file. The latest solidity compiler version, 0.8.17 , fixed important bugs in the compiler along with new native protections. The current version is missing the following fixes: 0.8.5, 0.8.6, 0.8.7, 0.8.8, 0.8.9, 0.8.12, 0.8.13, 0.8.14, 0.8.15, 0.8.16, 0.8.17.

The official Solidity recommendations are that you should use the latest released version of Solidity when deploying contracts. Apart from exceptional cases, only the most recent version receives security fixes.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to use the latest Solidity compiler version as possible.

Remediation Plan:

SOLVED: The Solidity compiler version was updated to 0.8.17.

Commit ID: 7e02156e7e6d133f8afc3a6b6f6fbd3ad3f84f82

3.6 (HAL-06) MISSING NATSPEC DOCUMENTATION - INFORMATIONAL

Description:

Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables and more. This special form is named the Ethereum Natural Language Specification Format (NatSpec).

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding documentation in Natspec format.

Remediation Plan:

SOLVED: Added Natspec documentation to the Root.sol contract.

Commit ID: 9fba0d4e1ff3c2e15e2a35a16b19c8ecb51892ca

3.7 (HAL-07) FUNCTIONS DO NOT FOLLOW NAMING CONVENTION - INFORMATIONAL

Description:

The min and max functions of the Root token contract do not follow naming conventions.
Both functions are defined internal but do not contain underscores in the name like other internal functions in the contract.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider renaming functions to follow the naming convention of other internal.

Remediation Plan:

SOLVED: Functions were renamed.

Commit ID: e5dde5ea911a9ecdd3e2e77c2ddde83ea625c7cc

3.8 (HAL-08) FUNCTIONS COULD BE DEFINED EXTERNAL - INFORMATIONAL

Description:

public functions that are never called by the contract should be declared external, and its immutable parameters should be placed in calldata to save gas.

Code location:

- Root.initialize(string,string)
- Root.addWhitelistToken(address)
- Root.removeWhitelistToken(address)

- Root.convertLambdaToLambda(address,uint32)
- Root.convertLambdasToLambdas(address[],uint32[])
- Root.bdvPerRoot()

- Root.earn()
- Root.mintWithTokenPermit(DepositTransfer[],To,address,uint256,uint256,uint8,by
- Root.mintWithTokensPermit(DepositTransfer[],To,address[],uint256[],uint256,uin
- Root.redeemWithFarmBalancePermit(DepositTransfer[],From,address,uint256,uint25
- Root.redeem(DepositTransfer[],From)

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider using external attribute for functions never called from the contract.

Remediation Plan:

SOLVED: Function definitions were updated from public to external.

Commit ID: 39cd2f0eff9f7c3e7b731fa3736a5c87a7e74faf

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contract in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contract in the repository and was able to compile it correctly into its ABI and binary format, Slither was run against the contract. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Slither results:

Listing 7

```
Root (contracts/tokens/Root.sol#25-393) is an upgradeable contract that does not protect its initiliaze functions: Root.initialize(string,string) (contracts/tokens/Root.sol#55-62). Anyone can delete the contract with: UUPSUpgradeable.upgradeTo(address) (node_modules/@openzeppelin/contracts-upgradeable-8/proxy/utils/UUPSUpgradeable.sol#72-75)UUPSUpgradeable.upgradeToAndCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable-8/proxy/utils/UUPSUpgradeable.sol#85-88)Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unprotected-upgradeable-contract

LibTransfer.sendToken(IERC20,uint256,address,LibTransfer.To) (contracts/libraries/Token/LibTransfer.sol#71-81) uses a dangerous strict equality:
        - amount == 0 (contracts/libraries/Token/LibTransfer.sol#77)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in Root._transferAndMint(DepositTransfer[],To) (contracts/tokens/Root.sol#256-284):
        External calls:
        - (shares,bdv,stalk,seed) = _transferDeposits(depositTransfers,true) (contracts/tokens/Root.sol#260-265)
```

```
                - IBeanstalk(BEANSTALK_ADDRESS).update(address(
↳ this)) (contracts/tokens/Root.sol#298)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
↳ transferDeposits(msg.sender,address(this),depositTransfers[i].
↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
↳ contracts/tokens/Root.sol#310-317)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
↳ transferDeposits(address(this),msg.sender,depositTransfers[i].
↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
↳ contracts/tokens/Root.sol#310-317)
        State variables written after the call(s):
        - _mint(address(this),shares) (contracts/tokens/Root.sol
↳ #269)
                - _totalSupply += amount (node_modules/
↳ @openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable
↳ .sol#267)
        - _mint(msg.sender,shares) (contracts/tokens/Root.sol#279)
                - _totalSupply += amount (node_modules/
↳ @openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable
↳ .sol#267)
Reentrancy in Root._transferAndRedeem(DepositTransfer[],From) (
↳ contracts/tokens/Root.sol#226-254):
        External calls:
        - (shares,bdv,stalk,seed) = _transferDeposits(
↳ depositTransfers,false) (contracts/tokens/Root.sol#230-235)
                - IBeanstalk(BEANSTALK_ADDRESS).update(address(
↳ this)) (contracts/tokens/Root.sol#298)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
↳ transferDeposits(msg.sender,address(this),depositTransfers[i].
↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
↳ contracts/tokens/Root.sol#310-317)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
↳ transferDeposits(address(this),msg.sender,depositTransfers[i].
↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
↳ contracts/tokens/Root.sol#310-317)
        - IBeanstalk(BEANSTALK_ADDRESS).transferTokenFrom(this,msg
↳ .sender,burnAddress,shares,mode,To.EXTERNAL) (contracts/tokens/
↳ Root.sol#242-249)
        State variables written after the call(s):
        - _burn(burnAddress,shares) (contracts/tokens/Root.sol
↳ #251)
                - _totalSupply -= amount (node_modules/
↳ @openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable
↳ .sol#295)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#reentrancy-vulnerabilities-1

Root._transferDeposits(DepositTransfer[],bool).j (contracts/tokens
↳ /Root.sol#318) is a local variable never initialized
Root.convertLambdasToLambdas(address[],uint32[]).i (contracts/
↳ tokens/Root.sol#93) is a local variable never initialized
Root._transferDeposits(DepositTransfer[],bool).i (contracts/tokens
↳ /Root.sol#304) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#uninitialized-local-variables

TokenFacet.getInternalBalances(address,IERC20[]).i (contracts/farm
↳ /facets/TokenFacet.sol#213) is a local variable never initialized
TokenFacet.getBalances(address,IERC20[]).i (contracts/farm/facets/
↳ TokenFacet.sol#255) is a local variable never initialized
SiloFacet.enrootDeposits(address,uint32[],uint256[]).i (contracts/
↳ farm/facets/SiloFacet/SiloFacet.sol#222) is a local variable never
↳  initialized
SiloFacet.permitDeposits(address,address,address[],uint256[],
↳ uint256,uint8,bytes32,bytes32).i (contracts/farm/facets/SiloFacet/
↳ SiloFacet.sol#159) is a local variable never initialized
TokenFacet.getExternalBalances(address,IERC20[]).i (contracts/farm
↳ /facets/TokenFacet.sol#234) is a local variable never initialized
TokenSilo._transferDeposits(address,address,address,uint32[],
↳ uint256[]).i (contracts/farm/facets/SiloFacet/TokenSilo.sol#334)
↳ is a local variable never initialized
TokenFacet.getAllBalances(address,IERC20[]).i (contracts/farm/
↳ facets/TokenFacet.sol#278) is a local variable never initialized
TokenSilo._claimWithdrawals(address,address,uint32[]).i (contracts
↳ /farm/facets/SiloFacet/TokenSilo.sol#274) is a local variable
↳ never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#uninitialized-local-variables

Root.initialize(string,string).name (contracts/tokens/Root.sol#55)
↳  shadows:
        - ERC20Upgradeable.name() (node_modules/@openzeppelin/
↳ contracts-upgradeable-8/token/ERC20/ERC20Upgradeable.sol#67-69) (
↳ function)
        - IERC20MetadataUpgradeable.name() (node_modules/
↳ @openzeppelin/contracts-upgradeable-8/token/ERC20/extensions/
↳ IERC20MetadataUpgradeable.sol#17) (function)
```

```
Root.initialize(string,string).symbol (contracts/tokens/Root.sol#55) shadows:
        - ERC20Upgradeable.symbol() (node_modules/@openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable.sol#75-77) (function)
        - IERC20MetadataUpgradeable.symbol() (node_modules/@openzeppelin/contracts-upgradeable-8/token/ERC20/extensions/IERC20MetadataUpgradeable.sol#22) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

SiloFacet.withdrawDeposit(address,uint32,uint256).season (contracts/farm/facets/SiloFacet/SiloFacet.sol#45) shadows:
        - SiloExit.season() (contracts/farm/facets/SiloFacet/SiloExit.sol#193-195) (function)
SiloFacet.claimWithdrawal(address,uint32,LibTransfer.To).season (contracts/farm/facets/SiloFacet/SiloFacet.sol#65) shadows:
        - SiloExit.season() (contracts/farm/facets/SiloFacet/SiloExit.sol#193-195) (function)
SiloFacet.transferDeposit(address,address,address,uint32,uint256).season (contracts/farm/facets/SiloFacet/SiloFacet.sol#89) shadows:
        - SiloExit.season() (contracts/farm/facets/SiloFacet/SiloExit.sol#193-195) (function)
SiloFacet.permitDeposits(address,address,address[],uint256[],uint256,uint8,bytes32,bytes32).s (contracts/farm/facets/SiloFacet/SiloFacet.sol#156) shadows:
        - ReentrancyGuard.s (contracts/farm/ReentrancyGuard.sol#17) (state variable)
SiloFacet.permitDeposit(address,address,address,uint256,uint256,uint8,bytes32,bytes32).s (contracts/farm/facets/SiloFacet/SiloFacet.sol#172) shadows:
        - ReentrancyGuard.s (contracts/farm/ReentrancyGuard.sol#17) (state variable)
SiloFacet.enrootDeposit(address,uint32,uint256)._season (contracts/farm/facets/SiloFacet/SiloFacet.sol#253) shadows:
        - TokenSilo._season() (contracts/farm/facets/SiloFacet/TokenSilo.sol#398-400) (function)
TokenSilo.getDeposit(address,address,uint32).season (contracts/farm/facets/SiloFacet/TokenSilo.sol#81) shadows:
        - SiloExit.season() (contracts/farm/facets/SiloFacet/SiloExit.sol#193-195) (function)
TokenSilo.getWithdrawal(address,address,uint32).season (contracts/farm/facets/SiloFacet/TokenSilo.sol#89) shadows:
        - SiloExit.season() (contracts/farm/facets/SiloFacet/
↳ SiloExit.sol#193-195) (function)
TokenSilo._withdrawDeposit(address,address,uint32,uint256).season
↳ (contracts/farm/facets/SiloFacet/TokenSilo.sol#164) shadows:
        - SiloExit.season() (contracts/farm/facets/SiloFacet/
↳ SiloExit.sol#193-195) (function)
TokenSilo.removeDeposit(address,address,uint32,uint256).season (
↳ contracts/farm/facets/SiloFacet/TokenSilo.sol#192) shadows:
        - SiloExit.season() (contracts/farm/facets/SiloFacet/
↳ SiloExit.sol#193-195) (function)
TokenSilo._claimWithdrawal(address,address,uint32).season (
↳ contracts/farm/facets/SiloFacet/TokenSilo.sol#259) shadows:
        - SiloExit.season() (contracts/farm/facets/SiloFacet/
↳ SiloExit.sol#193-195) (function)
TokenSilo._removeTokenWithdrawal(address,address,uint32).season (
↳ contracts/farm/facets/SiloFacet/TokenSilo.sol#289) shadows:
        - SiloExit.season() (contracts/farm/facets/SiloFacet/
↳ SiloExit.sol#193-195) (function)
TokenSilo._transferDeposit(address,address,address,uint32,uint256)
↳ .season (contracts/farm/facets/SiloFacet/TokenSilo.sol#306)
↳ shadows:
        - SiloExit.season() (contracts/farm/facets/SiloFacet/
↳ SiloExit.sol#193-195) (function)
TokenFacet.permitToken(address,address,address,uint256,uint256,
↳ uint8,bytes32,bytes32).s (contracts/farm/facets/TokenFacet.sol
↳ #157) shadows:
        - ReentrancyGuard.s (contracts/farm/ReentrancyGuard.sol
↳ #17) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#local-variable-shadowing

Root._convertLambdaToLambda(address,uint32) (contracts/tokens/Root
↳ .sol#98-110) has external calls inside a loop: (amount) =
↳ IBeanstalk(BEANSTALK_ADDRESS).getDeposit(address(this),token,
↳ season
) (contracts/tokens/Root.sol#99)
Root._convertLambdaToLambda(address,uint32) (contracts/tokens/Root
↳ .sol#98-110) has external calls inside a loop: (fromBdv,toBdv) =
↳ IBeanstalk(BEANSTALK_ADDRESS).convert(abi.encode(ConvertKind
.LAMBDA_LAMBDA,amount,token),seasons,amounts) (contracts/tokens/
↳ Root.sol#104-108)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation/#calls-inside-a-loop

```

```
TokenFacet.getExternalBalance(address,IERC20) (contracts/farm/
facets/TokenFacet.sol#220-226) has external calls inside a loop:
balance = token.balanceOf(account) (contracts/farm/facets/TokenF
acet.sol#225)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation/#calls-inside-a-loop

Reentrancy in Root._convertLambdaToLambda(address,uint32) (
contracts/tokens/Root.sol#98-110):
        External calls:
        - (fromBdv,toBdv) = IBeanstalk(BEANSTALK_ADDRESS).convert(
abi.encode(ConvertKind.LAMBDA_LAMBDA,amount,token),seasons,amounts
) (contracts/tokens/Root.sol#104-108)
        State variables written after the call(s):
        - underlyingBdv += toBdv - fromBdv (contracts/tokens/Root.
sol#109)
Reentrancy in Root._transferAndMint(DepositTransfer[],To) (
contracts/tokens/Root.sol#256-284):
        External calls:
        - (shares,bdv,stalk,seed) = _transferDeposits(
depositTransfers,true) (contracts/tokens/Root.sol#260-265)
                - IBeanstalk(BEANSTALK_ADDRESS).update(address(
this)) (contracts/tokens/Root.sol#298)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
transferDeposits(msg.sender,address(this),depositTransfers[i].
token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
contracts/
tokens/Root.sol#310-317)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
transferDeposits(address(this),msg.sender,depositTransfers[i].
token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
contracts/
tokens/Root.sol#310-317)
        State variables written after the call(s):
        - _approve(address(this),BEANSTALK_ADDRESS,shares) (
contracts/tokens/Root.sol#270)
                - _allowances[owner][spender] = amount (
node_modules/@openzeppelin/contracts-upgradeable-8/token/ERC20/
ERC20Upgradeable.sol#323)
        - _mint(address(this),shares) (contracts/tokens/Root.sol
#269)
                - _balances[account] += amount (node_modules/
@openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable
.sol#268)
```

```
        - _mint(msg.sender,shares) (contracts/tokens/Root.sol#279)
                - _balances[account] += amount (node_modules/@openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable.sol#268)
Reentrancy in Root._transferAndRedeem(DepositTransfer[],From) (contracts/tokens/Root.sol#226-254):
        External calls:
        - (shares,bdv,stalk,seed) = _transferDeposits(depositTransfers,false) (contracts/tokens/Root.sol#230-235)
                - IBeanstalk(BEANSTALK_ADDRESS).update(address(this)) (contracts/tokens/Root.sol#298)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).transferDeposits(msg.sender,address(this),depositTransfers[i].token,depositTransfers[i].seasons,depositTransfers[i].amounts) (contracts/
tokens/Root.sol#310-317)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).transferDeposits(address(this),msg.sender,depositTransfers[i].token,depositTransfers[i].seasons,depositTransfers[i].amounts) (contracts/
tokens/Root.sol#310-317)
        - IBeanstalk(BEANSTALK_ADDRESS).transferTokenFrom(this,msg.sender,burnAddress,shares,mode,To.EXTERNAL) (contracts/tokens/Root.sol#242-249)
        State variables written after the call(s):
        - _burn(burnAddress,shares) (contracts/tokens/Root.sol#251)
                - _balances[account] = accountBalance - amount (node_modules/@openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable.sol#293)
Reentrancy in Root._transferDeposits(DepositTransfer[],bool) (contracts/tokens/Root.sol#286-392):
        External calls:
        - IBeanstalk(BEANSTALK_ADDRESS).update(address(this)) (contracts/tokens/Root.sol#298)
        State variables written after the call(s):
        - underlyingBdv = underlyingBdvAfter (contracts/tokens/Root.sol#391)
Reentrancy in Root.earn() (contracts/tokens/Root.sol#116-119):
        External calls:
        - beans = IBeanstalk(BEANSTALK_ADDRESS).plant() (contracts/tokens/Root.sol#117)
        State variables written after the call(s):
        - underlyingBdv += beans (contracts/tokens/Root.sol#118)
```

```
Reentrancy in Root.mintWithTokenPermit(DepositTransfer[],To,
 ↳ address,uint256,uint256,uint8,bytes32,bytes32) (contracts/tokens/
 ↳ Root.sol#139-161):
        External calls:
        - IBeanstalk(BEANSTALK_ADDRESS).permitDeposit(msg.sender,
 ↳ address(this),token,value,deadline,v,r,s) (contracts/tokens/Root.
 ↳ sol#149-158)
        - _transferAndMint(depositTransfers,mode) (contracts/
 ↳ tokens/Root.sol#160)
                - IBeanstalk(BEANSTALK_ADDRESS).update(address(
 ↳ this)) (contracts/tokens/Root.sol#298)
                - IBeanstalk(BEANSTALK_ADDRESS).transferToken(this
 ↳ ,msg.sender,shares,From.EXTERNAL,To.INTERNAL) (contracts/tokens/
 ↳ Root.sol#271-277)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
 ↳ transferDeposits(msg.sender,address(this),depositTransfers[i].
 ↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
 ↳ contracts/
tokens/Root.sol#310-317)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
 ↳ transferDeposits(address(this),msg.sender,depositTransfers[i].
 ↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
 ↳ contracts/
tokens/Root.sol#310-317)
        State variables written after the call(s):
        - _transferAndMint(depositTransfers,mode) (contracts/
 ↳ tokens/Root.sol#160)
                - _allowances[owner][spender] = amount (
 ↳ node_modules/@openzeppelin/contracts-upgradeable-8/token/ERC20/
 ↳ ERC20Upgradeable.sol#323)
Reentrancy in Root.mintWithTokensPermit(DepositTransfer[],To,
 ↳ address[],uint256[],uint256,uint8,bytes32,bytes32) (contracts/
 ↳ tokens/Root.sol#163-185):
        External calls:
        - IBeanstalk(BEANSTALK_ADDRESS).permitDeposits(msg.sender,
 ↳ address(this),tokens,values,deadline,v,r,s) (contracts/tokens/Root
 ↳ .sol#173-182)
        - _transferAndMint(depositTransfers,mode) (contracts/
 ↳ tokens/Root.sol#184)
                - IBeanstalk(BEANSTALK_ADDRESS).update(address(
 ↳ this)) (contracts/tokens/Root.sol#298)
                - IBeanstalk(BEANSTALK_ADDRESS).transferToken(this
 ↳ ,msg.sender,shares,From.EXTERNAL,To.INTERNAL) (contracts/tokens/
 ↳ Root.sol#271-277)
```

```
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
 ↳ transferDeposits(msg.sender,address(this),depositTransfers[i].
 ↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
 ↳ contracts/
tokens/Root.sol#310-317)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
 ↳ transferDeposits(address(this),msg.sender,depositTransfers[i].
 ↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
 ↳ contracts/
tokens/Root.sol#310-317)
        State variables written after the call(s):
        - _transferAndMint(depositTransfers,mode) (contracts/
 ↳ tokens/Root.sol#184)
                - _allowances[owner][spender] = amount (
 ↳ node_modules/@openzeppelin/contracts-upgradeable-8/token/ERC20/
 ↳ ERC20Upgradeable.sol#323)
Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#reentrancy-vulnerabilities-2

Reentrancy in Root._transferAndMint(DepositTransfer[],To) (
 ↳ contracts/tokens/Root.sol#256-284):
        External calls:
        - (shares,bdv,stalk,seed) = _transferDeposits(
 ↳ depositTransfers,true) (contracts/tokens/Root.sol#260-265)
                - IBeanstalk(BEANSTALK_ADDRESS).update(address(
 ↳ this)) (contracts/tokens/Root.sol#298)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
 ↳ transferDeposits(msg.sender,address(this),depositTransfers[i].
 ↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
 ↳ contracts/
tokens/Root.sol#310-317)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
 ↳ transferDeposits(address(this),msg.sender,depositTransfers[i].
 ↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
 ↳ contracts/
tokens/Root.sol#310-317)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (node_modules/
 ↳ @openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable
 ↳ .sol#324)
                - _approve(address(this),BEANSTALK_ADDRESS,shares)
 ↳  (contracts/tokens/Root.sol#270)
        - Transfer(address(0),account,amount) (node_modules/
 ↳ @openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable
↳ .sol#269)
                  - _mint(address(this),shares) (contracts/tokens/
↳ Root.sol#269)
          - Transfer(address(0),account,amount) (node_modules/
↳ @openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable
↳ .sol#269)
                  - _mint(msg.sender,shares) (contracts/tokens/Root.
↳ sol#279)
Reentrancy in Root._transferAndMint(DepositTransfer[],To) (
↳ contracts/tokens/Root.sol#256-284):
        External calls:
        - (shares,bdv,stalk,seed) = _transferDeposits(
↳ depositTransfers,true) (contracts/tokens/Root.sol#260-265)
                  - IBeanstalk(BEANSTALK_ADDRESS).update(address(
↳ this)) (contracts/tokens/Root.sol#298)
                  - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
↳ transferDeposits(msg.sender,address(this),depositTransfers[i].
↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
↳ contracts/
tokens/Root.sol#310-317)
                  - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
↳ transferDeposits(address(this),msg.sender,depositTransfers[i].
↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
↳ contracts/
tokens/Root.sol#310-317)
          - IBeanstalk(BEANSTALK_ADDRESS).transferToken(this,msg.
↳ sender,shares,From.EXTERNAL,To.INTERNAL) (contracts/tokens/Root.
↳ sol#271-277)
        Event emitted after the call(s):
          - Mint(msg.sender,depositTransfers,bdv,stalk,seed,shares)
↳ (contracts/tokens/Root.sol#282)
Reentrancy in Root._transferAndRedeem(DepositTransfer[],From) (
↳ contracts/tokens/Root.sol#226-254):
        External calls:
          - (shares,bdv,stalk,seed) = _transferDeposits(
↳ depositTransfers,false) (contracts/tokens/Root.sol#230-235)
                  - IBeanstalk(BEANSTALK_ADDRESS).update(address(
↳ this)) (contracts/tokens/Root.sol#298)
                  - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
↳ transferDeposits(msg.sender,address(this),depositTransfers[i].
↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
↳ contracts/
tokens/Root.sol#310-317)
```

```
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
↳ transferDeposits(address(this),msg.sender,depositTransfers[i].
↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
↳ contracts/
tokens/Root.sol#310-317)
        - IBeanstalk(BEANSTALK_ADDRESS).transferTokenFrom(this,msg
↳ .sender,burnAddress,shares,mode,To.EXTERNAL) (contracts/tokens/
↳ Root.sol#242-249)
        Event emitted after the call(s):
        - Redeem(msg.sender,depositTransfers,bdv,stalk,seed,shares
↳ ) (contracts/tokens/Root.sol#252)
        - Transfer(account,address(0),amount) (node_modules/
↳ @openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable
↳ .sol#297)
                - _burn(burnAddress,shares) (contracts/tokens/Root
↳ .sol#251)
Reentrancy in Root.mintWithTokenPermit(DepositTransfer[],To,
↳ address,uint256,uint256,uint8,bytes32,bytes32) (contracts/tokens/
↳ Root.sol#139-161):
        External calls:
        - IBeanstalk(BEANSTALK_ADDRESS).permitDeposit(msg.sender,
↳ address(this),token,value,deadline,v,r,s) (contracts/tokens/Root.
↳ sol#149-158)
        - _transferAndMint(depositTransfers,mode) (contracts/
↳ tokens/Root.sol#160)
                - IBeanstalk(BEANSTALK_ADDRESS).update(address(
↳ this)) (contracts/tokens/Root.sol#298)
                - IBeanstalk(BEANSTALK_ADDRESS).transferToken(this
↳ ,msg.sender,shares,From.EXTERNAL,To.INTERNAL) (contracts/tokens/
↳ Root.sol#271-277)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
↳ transferDeposits(msg.sender,address(this),depositTransfers[i].
↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
↳ contracts/
tokens/Root.sol#310-317)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
↳ transferDeposits(address(this),msg.sender,depositTransfers[i].
↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
↳ contracts/
tokens/Root.sol#310-317)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (node_modules/
↳ @openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable
↳ .sol#324)
```

```
                - _transferAndMint(depositTransfers,mode) (
 ↳ contracts/tokens/Root.sol#160)
        - Mint(msg.sender,depositTransfers,bdv,stalk,seed,shares)
 ↳ (contracts/tokens/Root.sol#282)
                - _transferAndMint(depositTransfers,mode) (
 ↳ contracts/tokens/Root.sol#160)
        - Transfer(address(0),account,amount) (node_modules/
 ↳ @openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable
 ↳ .sol#269)
                - _transferAndMint(depositTransfers,mode) (
 ↳ contracts/tokens/Root.sol#160)
Reentrancy in Root.mintWithTokensPermit(DepositTransfer[],To,
 ↳ address[],uint256[],uint256,uint8,bytes32,bytes32) (contracts/
 ↳ tokens/Root.sol#163-185):
        External calls:
        - IBeanstalk(BEANSTALK_ADDRESS).permitDeposits(msg.sender,
 ↳ address(this),tokens,values,deadline,v,r,s) (contracts/tokens/Root
 ↳ .sol#173-182)
        - _transferAndMint(depositTransfers,mode) (contracts/
 ↳ tokens/Root.sol#184)
                - IBeanstalk(BEANSTALK_ADDRESS).update(address(
 ↳ this)) (contracts/tokens/Root.sol#298)
                - IBeanstalk(BEANSTALK_ADDRESS).transferToken(this
 ↳ ,msg.sender,shares,From.EXTERNAL,To.INTERNAL) (contracts/tokens/
 ↳ Root.sol#271-277)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
 ↳ transferDeposits(msg.sender,address(this),depositTransfers[i].
 ↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
 ↳ contracts/
tokens/Root.sol#310-317)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
 ↳ transferDeposits(address(this),msg.sender,depositTransfers[i].
 ↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
 ↳ contracts/
tokens/Root.sol#310-317)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (node_modules/
 ↳ @openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable
 ↳ .sol#324)
                - _transferAndMint(depositTransfers,mode) (
 ↳ contracts/tokens/Root.sol#184)
        - Mint(msg.sender,depositTransfers,bdv,stalk,seed,shares)
 ↳ (contracts/tokens/Root.sol#282)
```

```
                - _transferAndMint(depositTransfers,mode) (
 ↳ contracts/tokens/Root.sol#184)
        - Transfer(address(0),account,amount) (node_modules/
 ↳ @openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable
 ↳ .sol#269)
                - _transferAndMint(depositTransfers,mode) (
 ↳ contracts/tokens/Root.sol#184)
Reentrancy in Root.redeemWithFarmBalancePermit(DepositTransfer[],
 ↳ From,address,uint256,uint256,uint8,bytes32,bytes32) (contracts/
 ↳ tokens/Root.sol#195-216):
        External calls:
        - IBeanstalk(BEANSTALK_ADDRESS).permitToken(msg.sender,
 ↳ address(this),token,value,deadline,v,r,s) (contracts/tokens/Root.
 ↳ sol#205-214)
        - _transferAndRedeem(depositTransfers,mode) (contracts/
 ↳ tokens/Root.sol#215)
                - IBeanstalk(BEANSTALK_ADDRESS).update(address(
 ↳ this)) (contracts/tokens/Root.sol#298)
                - IBeanstalk(BEANSTALK_ADDRESS).transferTokenFrom(
 ↳ this,msg.sender,burnAddress,shares,mode,To.EXTERNAL) (contracts/
 ↳ tokens/Root.sol#242-249)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
 ↳ transferDeposits(msg.sender,address(this),depositTransfers[i].
 ↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
 ↳ contracts/
tokens/Root.sol#310-317)
                - bdvs = IBeanstalk(BEANSTALK_ADDRESS).
 ↳ transferDeposits(address(this),msg.sender,depositTransfers[i].
 ↳ token,depositTransfers[i].seasons,depositTransfers[i].amounts) (
 ↳ contracts/
tokens/Root.sol#310-317)
        Event emitted after the call(s):
        - Redeem(msg.sender,depositTransfers,bdv,stalk,seed,shares
 ↳ ) (contracts/tokens/Root.sol#252)
                - _transferAndRedeem(depositTransfers,mode) (
 ↳ contracts/tokens/Root.sol#215)
        - Transfer(account,address(0),amount) (node_modules/
 ↳ @openzeppelin/contracts-upgradeable-8/token/ERC20/ERC20Upgradeable
 ↳ .sol#297)
                - _transferAndRedeem(depositTransfers,mode) (
 ↳ contracts/tokens/Root.sol#215)
Reference: https://github.com/crytic/slither/wiki/Detector-
 ↳ Documentation#reentrancy-vulnerabilities-3

```

```
LibSiloPermit.permit(address,address,address,uint256,uint256,uint8
↳ ,bytes32,bytes32) (contracts/libraries/Silo/LibSiloPermit.sol
↳ #25-40) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,Silo:
↳ permit expired deadline) (contracts/libraries/Silo/LibSiloPermit.
↳ sol#35)
LibSiloPermit.permits(address,address,address[],uint256[],uint256,
↳ uint8,bytes32,bytes32) (contracts/libraries/Silo/LibSiloPermit.sol
↳ #42-57) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,Silo:
↳ permit expired deadline) (contracts/libraries/Silo/LibSiloPermit.
↳ sol#52)
LibTokenPermit.permit(address,address,address,uint256,uint256,
↳ uint8,bytes32,bytes32) (contracts/libraries/Token/LibTokenPermit.
↳ sol#24-39) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,Token:
↳ permit expired deadline) (contracts/libraries/Token/LibTokenPermit
↳ .sol#34)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#block-timestamp

Different versions of Solidity are used:
        - Version used: ['>=0.4.22<0.9.0', '>=0.7.5', '^0.8.0',
↳ '^0.8.1', '^0.8.2']
        - ^0.8.0 (contracts/interfaces/IBeanstalk.sol#2)
        - ABIEncoderV2 (contracts/interfaces/IBeanstalk.sol#3)
        - ^0.8.0 (contracts/interfaces/IDelegation.sol#2)
        - ABIEncoderV2 (contracts/interfaces/IDelegation.sol#3)
        - >=0.7.5 (contracts/interfaces/IQuoter.sol#2)
        - v2 (contracts/interfaces/IQuoter.sol#3)
        - >=0.7.5 (contracts/interfaces/ISwapRouter.sol#2)
        - v2 (contracts/interfaces/ISwapRouter.sol#3)
        - ^0.8.0 (contracts/tokens/Root.sol#5)
        - ABIEncoderV2 (contracts/tokens/Root.sol#6)
        - >=0.4.22<0.9.0 (node_modules/hardhat/console.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#different-pragma-directives-are-used

Root._convertLambdaToLambda(address,uint32) (contracts/tokens/Root
↳ .sol#98-110) has costly operations inside a loop:
        - underlyingBdv += toBdv - fromBdv (contracts/tokens/Root.sol#109)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop

Pragma version^0.8.0 (contracts/interfaces/IBeanstalk.sol#2) allows old versions
Pragma version^0.8.0 (contracts/interfaces/IDelegation.sol#2) allows old versions
Pragma version^0.8.0 (contracts/tokens/Root.sol#5) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter Root.setDelegate(address,address,bytes32)._delegateContract (contracts/tokens/Root.sol#77) is not in mixedCase
Parameter Root.setDelegate(address,address,bytes32)._delegate (contracts/tokens/Root.sol#78) is not in mixedCase
Parameter Root.setDelegate(address,address,bytes32)._snapshotId (contracts/tokens/Root.sol#79) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

initialize(string,string) should be declared external:
        - Root.initialize(string,string) (contracts/tokens/Root.sol#55-62)
addWhitelistToken(address) should be declared external:
        - Root.addWhitelistToken(address) (contracts/tokens/Root.sol#66-69)
removeWhitelistToken(address) should be declared external:
        - Root.removeWhitelistToken(address) (contracts/tokens/Root.sol#71-74)
convertLambdaToLambda(address,uint32) should be declared external:
        - Root.convertLambdaToLambda(address,uint32) (contracts/tokens/Root.sol#88-90)
convertLambdasToLambdas(address[],uint32[]) should be declared external:
        - Root.convertLambdasToLambdas(address[],uint32[]) (contracts/tokens/Root.sol#92-96)
bdvPerRoot() should be declared external:
        - Root.bdvPerRoot() (contracts/tokens/Root.sol#112-114)
earn() should be declared external:
        - Root.earn() (contracts/tokens/Root.sol#116-119)
```

```
mintWithTokenPermit(DepositTransfer[],To,address,uint256,uint256,uint8,bytes32,bytes32) should be declared external:
        - Root.mintWithTokenPermit(DepositTransfer[],To,address,uint256,uint256,uint8,bytes32,bytes32) (contracts/tokens/Root.sol#139-161)
mintWithTokensPermit(DepositTransfer[],To,address[],uint256[],uint256,uint8,bytes32,bytes32) should be declared external:
        - Root.mintWithTokensPermit(DepositTransfer[],To,address[],uint256[],uint256,uint8,bytes32,bytes32) (contracts/tokens/Root.sol#163-185)
mint(DepositTransfer[],To) should be declared external:
        - Root.mint(DepositTransfer[],To) (contracts/tokens/Root.sol#187-193)
redeemWithFarmBalancePermit(DepositTransfer[],From,address,uint256,uint256,uint8,bytes32,bytes32) should be declared external:
        - Root.redeemWithFarmBalancePermit(DepositTransfer[],From,address,uint256,uint256,uint8,bytes32,bytes32) (contracts/tokens/Root.sol#195-216)
redeem(DepositTransfer[],From) should be declared external:
        - Root.redeem(DepositTransfer[],From) (contracts/tokens/Root.sol#218-224)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Slither correctly flagged:
- Root token implementation can be uninitialized
- some functions can be defined external

Those issues are included in the findings section of the report.

No major issues found by Slither.

THANK YOU FOR CHOOSING

HALBORN