Seeder Finance Core

Smart Contract Audit Report
Prepared for Seeder Finance

Date Issued:	Jun 30, 2021
Project ID:	AUDIT2021005
Version:	v1.0
Confidentiality Level:	Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2021005
Version	v1.0
Client	Seeder Finance
Project	Seeder Finance Core
Auditor(s)	Weerawat Pawanawiwat Pongsakorn Sommalai Suvicha Buakhom
Author	Weerawat Pawanawiwat
Reviewer	Pongsakorn Sommalai
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Jun 30, 2021	Full report	Weerawat Pawanawiwat

Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	t.me/inspexco
Email	audit@inspex.co

Table of Contents

1. Executive Summary

As requested by Seeder Finance, Inspex team conducted an audit to verify the security posture of the Seeder Finance Core smart contracts between Jun 15, 2021 and Jun 17, 2021. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Seeder Finance Core smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found, and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Inspex found 3 critical, 3 high, 4 medium, 3 low, and 3 info-severity issues. With the project team's prompt response, 3 critical, 3 high, 4 medium, 2 low, and 3 info-severity issues were resolved or mitigated in the reassessment, while 1 low-severity issue was acknowledged by the team. Therefore, Inspex trusts that Seeder Finance Core smart contracts have sufficient protections to be safe for public use. However, in the long run, Inspex suggests resolving all issues found in this report.

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inpex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

Seeder Finance is a yield farming protocol on Binance Smart Chain. On Seeder Finance, users can seed their funds, plant the seeds into different farms, and harvest their earnings.

Scope Information:

Project Name	Seeder Finance Core
Website	https://seeder.finance/farms
Smart Contract Type	Ethereum Smart Contract
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Jun 15, 2021 - Jun 17, 2021
Reassessment Date	Jun 29, 2021

2.2. Scope

The following smart contracts were audited and reassessed by Inspex in detail:

Initial Audit (Commit: 6ef6a79b066c79138860682f96da71f0979c8cd9):

Smart Contract	File Name	Location (URL)
BigFarm	BigFarm.sol	https://github.com/seeder-finance/seeder-core-contract/blob/6ef6a79b066c79138860682f96da71f0979c8cd9/contracts/farming/BigFarm.sol
Granary	GranaryV2.sol	https://github.com/seeder-finance/seeder-core-contract/blob/6ef6a79b066c79138860682f96da71f0979c8cd9/contracts/farming/GranaryV2.sol
GreenHouse	GreenHouse.sol	https://github.com/seeder-finance/seeder-core-contract/blob/6ef6a79b066c79138860682f96da71f0979c8cd9/contracts/farming/GreenHouse.sol
Bank	BankV1_5.sol	https://github.com/seeder-finance/seeder-core-contract/blob/6ef6a79b066c79138860682f96da71f0979c8cd9/contracts/finance/BankV1_5.sol
Leaf	Leaf.sol	https://github.com/seeder-finance/seeder-core-contract/blob/6

		ef6a79b066c79138860682f96da71f0979c8cd9/contracts/tokens/Leaf.sol
sdToken	sdToken.sol	https://github.com/seeder-finance/seeder-core-contract/blob/6ef6a79b066c79138860682f96da71f0979c8cd9/contracts/tokens/sdToken.sol
Tree	Tree.sol	https://github.com/seeder-finance/seeder-core-contract/blob/6ef6a79b066c79138860682f96da71f0979c8cd9/contracts/tokens/Tree.sol
Loanable	Loanable.sol	https://github.com/seeder-finance/seeder-core-contract/blob/6ef6a79b066c79138860682f96da71f0979c8cd9/contracts/utils/Loanable.sol
TrustCaller	TrustCaller.sol	https://github.com/seeder-finance/seeder-core-contract/blob/6ef6a79b066c79138860682f96da71f0979c8cd9/contracts/utils/TrustCaller.sol

Reassessment (Commit: 28b4b9c340753aafff8ced39bdc3817ed4547dea):

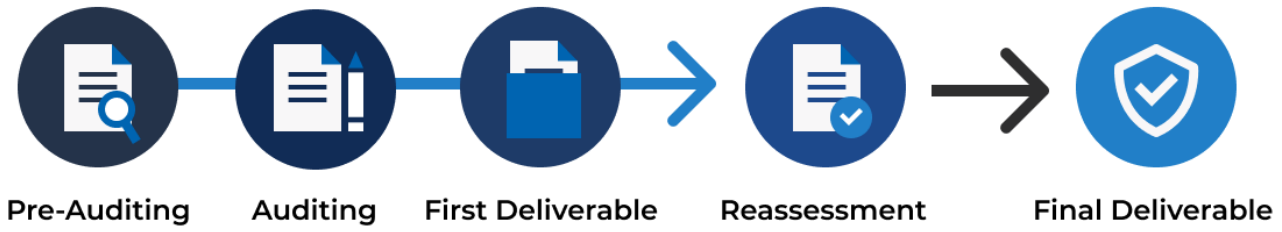
Smart Contract	File Name	Location (URL)
BigFarm	BigFarmV3.sol	https://github.com/seeder-finance/seeder-core-contract/blob/28b4b9c340753aafff8ced39bdc3817ed4547dea/contracts/V3/farming/BigFarmV3.sol
Granary	GranaryV3.sol	https://github.com/seeder-finance/seeder-core-contract/blob/28b4b9c340753aafff8ced39bdc3817ed4547dea/contracts/V3/farming/GranaryV3.sol
GreenHouse	GreenHouseV3.sol	https://github.com/seeder-finance/seeder-core-contract/blob/28b4b9c340753aafff8ced39bdc3817ed4547dea/contracts/V3/farming/GreenHouseV3.sol
Bank	BankV3.sol	https://github.com/seeder-finance/seeder-core-contract/blob/28b4b9c340753aafff8ced39bdc3817ed4547dea/contracts/V3/finance/BankV3.sol
Leaf	Leaf.sol	https://github.com/seeder-finance/seeder-core-contract/blob/28b4b9c340753aafff8ced39bdc3817ed4547dea/contracts/tokens/Leaf.sol
sdToken	sdToken.sol	https://github.com/seeder-finance/seeder-core-contract/blob/28b4b9c340753aafff8ced39bdc3817ed4547dea/contracts/tokens/sdToken.sol
Tree	Tree.sol	https://github.com/seeder-finance/seeder-core-contract/blob/28b4b9c340753aafff8ced39bdc3817ed4547dea/contracts/tokens

		/Tree.sol
Loanable	LoanableV3.sol	https://github.com/seeder-finance/seeder-core-contract/blob/28b4b9c340753aafff8ced39bdc3817ed4547dea/contracts/V3/utils/LoanableV3.sol
TrustCaller	TrustCaller.sol	https://github.com/seeder-finance/seeder-core-contract/blob/28b4b9c340753aafff8ced39bdc3817ed4547dea/contracts/utils/TrustCaller.sol

3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue



3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The following audit items were checked during the auditing activity.

General
Reentrancy Attack
Integer Overflows and Underflows
Unchecked Return Values for Low-Level Calls
Bad Randomness
Transaction Ordering Dependence
Time Manipulation
Short Address Attack
Outdated Compiler Version
Use of Known Vulnerable Component
Deprecated Solidity Features
Use of Deprecated Component
Loop with High Gas Consumption
Unauthorized Self-destruct
Redundant Fallback Function
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication
Upgradable Without Timelock
Improper Kill-Switch Mechanism
Improper Front-end Integration
Insecure Smart Contract Initiation

Denial of Service
Improper Oracle Usage
Memory Corruption
Best Practice
Use of Variadic Byte Array
Implicit Compiler Version
Implicit Visibility Level
Implicit Type Inference
Function Declaration Inconsistency
Token API Violation
Best Practices Violation

3.3. Risk Rating

OWASP Risk Rating Methodology[1] is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker
- **Impact**: a measure of the damage caused by a successful attack

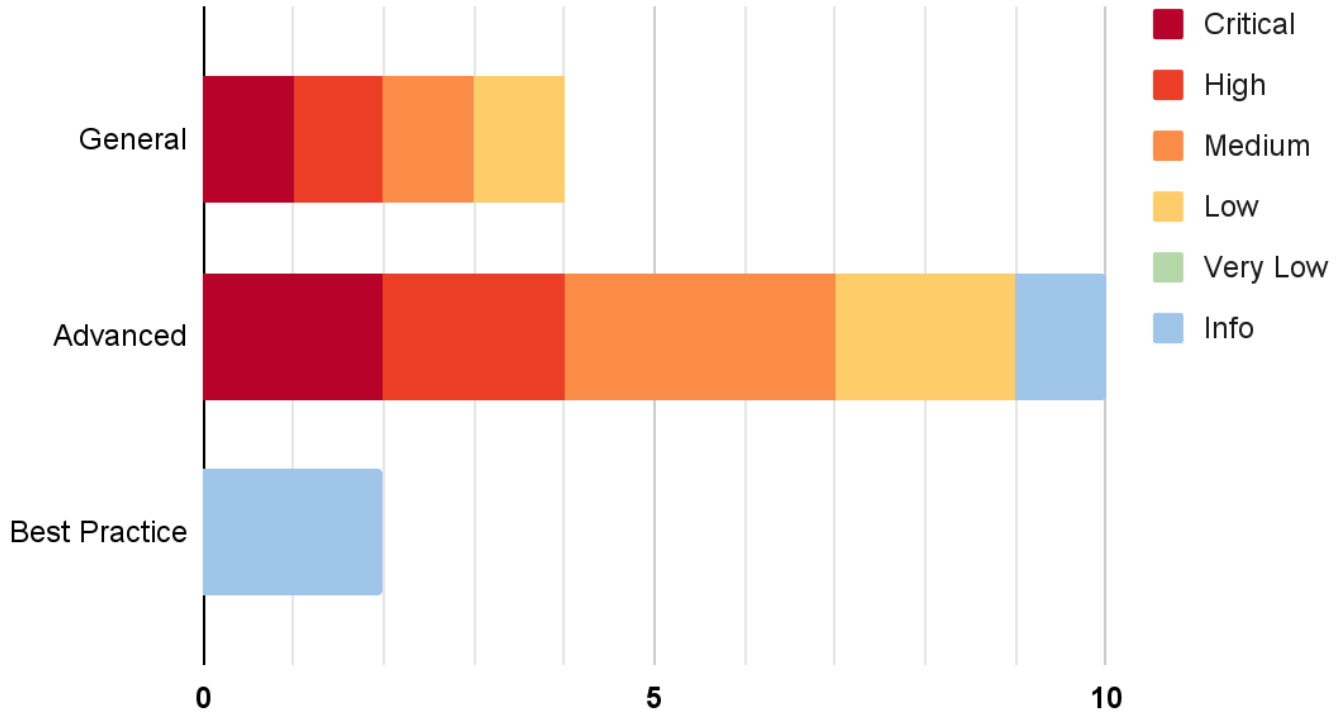
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Impact \ Likelihood	Low	Medium	High
Low	**Very Low**	**Low**	**Medium**
Medium	**Low**	**Medium**	**High**
High	**Medium**	**High**	**Critical**

4. Summary of Findings

From the assessments, Inspex has found 16 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complication.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

The information and status of each issue can be found in the following table:

ID	Issue	Category	Severity	Status
IDX-001	Upgradable Proxy Contract Without Timelock	Advanced	Critical	Resolved
IDX-002	Token Draining via Adding Duplicated ibToken Address	Advanced	Critical	Resolved
IDX-003	Token Manual Minting by Contract Owner	General	Critical	Resolved *
IDX-004	Denial of Services in BigFarm	Advanced	High	Resolved
IDX-005	Centralized Control of State Variable	General	High	Resolved
IDX-006	Improper Share Calculation	Advanced	High	Resolved
IDX-007	Incorrect Token Distribution	Advanced	Medium	Resolved
IDX-008	Improper Reward Calculation (1)	Advanced	Medium	Resolved *
IDX-009	Unsafe Token Transfer	General	Medium	Resolved
IDX-010	Inconsistent Token Transfer Fee	Advanced	Medium	Resolved *
IDX-011	Design Flaw in updateAllFarms() Function	General	Low	Acknowledged
IDX-012	Improper Reward Calculation (2)	Advanced	Low	Resolved
IDX-013	Improper Release of Locked Tokens	Advanced	Low	Resolved
IDX-014	Inexplicit Solidity Compiler Version	Best Practice	Info	Resolved
IDX-015	Improper Function Visibility	Best Practice	Info	Resolved
IDX-016	Design Flaw in the Loanable Library	Advanced	Info	Resolved

* The mitigations or clarifications by Seeder Finance can be found in Chapter 5.

5. Detailed Findings Information

5.1. Upgradable Proxy Contract Without Timelock

ID	IDX-001
Target	BigFarm.sol GranaryV2.sol GreenHouse.sol BankV1_5.sol
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: Critical** **Impact: High** The contract owner can deploy new logic to the proxy. **Likelihood: High** It is very likely that the logic of affected contracts can be changed anytime without being notified by users.
Status	Resolved Seeder Finance team has resolved this issue by implementing a timelock over the `ProxyAdmin` contract at the following address: 0xc5935ef3d390fde99dc57cc768a674b829e17805 The `Timelock` contract can be found at the following address: 0xbe9E4627881D03940a1E1B64c2D7fFc62813f5a5

5.1.1. Description

At the time of the audit (block `8401419`), the upgradable smart contracts were deployed without a timelock mechanism.

Name	Contract Address	Owner of Proxy Admin
BigFarm	0x1aF28E7b1A03fA107961897a28449F4F9768ac75	0x21f94A5217eCAedd9092ce88de598C7d526638cA
GranaryV2	0xbf5B3409a8942168283ec7f3e065f199D6Fef195	0x21f94A5217eCAedd9092ce88de598C7d526638cA
BankV1_5	0x99dD1c7a2893931D209fA5C57FE65f34d4C11db8	0x21f94A5217eCAedd9092ce88de598C7d526638cA

GreenHouse	0xbf5B3409a8942168283ec7f3e065f199D6Fef195	0x21f94A5217eCAedd9092ce88de598C7d526638cA

Please note that the address (0x21f94A5217eCAedd9092ce88de598C7d526638cA) is not the `Timelock` contract.

As a result, the contract owner can deploy new logic to the proxy anytime without notifying the users.

5.1.2. Remediation

Inspex recommends performing the following actions:

- Deploy the `Timelock` contract
- Set the minimum delay to at least 24 hours
- Set the owner of proxy contract to the `Timelock` contract

5.2. Token Draining via Adding Duplicated ibToken Address

ID	IDX-002
Target	BankV1_5.sol
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: Critical** **Impact: High** The tokens deposited to the contract can be drained by the owner of the contract. **Likelihood: High** There is currently no restriction to prevent the owner from performing this attack.
Status	Resolved Seeder Finance team has resolved this issue in `BankV3.sol`.

5.2.1. Description

Token pairs can be added to the `BankV1_5` contract by the contract owner using the `addDepositPair()` function. However, there is no checking done to make sure that the `ibToken` has never been added before.

BankV1_5.sol

```
307 function addDepositPair(address originTokenAddress, address ibTokenAddress)
    external onlyOwner {
308     require(originTokenAddress != address(0), "Cannot add zero address in the
    pairs");
309     require(ibTokenAddress != address(0), "Cannot add zero address in the
    pairs");
310     require(address(_ibNativeToken) != ibTokenAddress, "Cannot add existing
    ibToken");
311
312     DepositPair storage depositPair = _tokenDepositPairs[originTokenAddress];
313     require(address(depositPair.originToken) == address(0), "Cannot add
    existing ibToken");
314
315     depositPair.originToken = IERC20(originTokenAddress);
316     depositPair.ibToken = IERC20(ibTokenAddress);
317
318     _supportOriginTokens.push(originTokenAddress);
319 }
```

This allows the owner to add any custom token to the `BankV1_5` contract together with an existing `ibToken`. Using this pair, the owner can mint a large amount of custom token, deposit the custom token to this pool to get the `ibToken`, and use the `ibToken` to withdraw other tokens with value.

5.2.2. Remediation

Inspex suggests checking the `ibTokenAddress` parameter for `addDepositPair()` function in Bank contract to make sure that there is no existing pair with the same `ibToken`.

5.3. Token Manual Minting by Contract Owner

ID	IDX-003
Target	Leaf.sol Tree.sol sdToken.sol
Category	General Smart Contract Vulnerability
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity: Critical** **Impact: High** The contract owner can arbitrarily mint the affected tokens. **Likelihood: High** It is very likely that the contract owner can set his wallet address to be the `TrustCaller` and call the `mint()` function.
Status	Resolved * The owner of $LEAF and `sdToken` has been renounced. For $TREE, there are business cases to customize the `TrustCaller`. However, the `Timelock` mechanism has already been applied. Inspex suggests that the users should frequently monitor the `TrustCaller` of $TREE. The renouncing of ownership is done in the following transactions on BSC: - 0x9a71fff011ea88eb2f3e156f43edca8c7fb83ea9219b35ffe0b819a9f6f9c74d - 0x636ad1074404803838d51cede45ab4151c74b0bb24545eba2be18b22b698cd11 - 0x8087aa48160bc24a149de25b67d6084d11ad70624ce50dfbc3c73cd99ee929b4

5.3.1. Description

In the $LEAF, $TREE, and `sdToken` contracts, the `mint()` function has `onlyTrustCaller` as a modifier as shown below.

Leaf.sol

```
21 function mint(address account, uint256 amount) external onlyTrustCaller {
22     require(totalSupply().add(amount) <= LIMIT_TOTAL_SUPPLY, "Cannot more than
   limit");
23
24     _mint(account, amount);
25 }
```

Tree.sol

```
25 function mint(address account, uint256 amount) external onlyTrustCaller {
```

```
26      require(account != address(0), "Tree: Minting to the zero address");
27
28      _mint(account, amount);
29  }
```

sdToken.sol

```
12  function mint(address account, uint256 amount) external onlyTrustCaller {
13      _mint(account, amount);
14  }
```

The `onlyTrustCaller` modifier checks that the caller is trusted by verifying the `msg.sender` with `_trustCallers` mapping.

TrustCaller.sol

```
25  modifier onlyTrustCaller() {
26      require(_trustCallers[msg.sender], "Caller is not trust");
27      _;
28  }
```

The contract owner can set any address to be the trusted caller by calling the `setTrustCaller()` function.

TrustCaller.sol

```
19  function setTrustCaller(address callerAddress, bool isTrusted) external
    onlyOwner {
20      _trustCallers[callerAddress] = isTrusted;
21
22      emit TrustCallerSet(callerAddress, isTrusted);
23  }
```

As a result, although the contract owner is currently not set to be the trusted caller, the owner will still be able to set the owner's wallet address as the trusted caller and call the `mint()` function in order to mint the affected tokens.

5.3.2. Remediation

Inspex suggests removing the `TrustCaller` library from the token contract, setting the `onlyOwner` as the modifier of `mint()` function, and setting the owner of the tokens to be the `BigFarm` contract.

However, Seeder Finance has already deployed the token contracts to the BSC mainnet. To fix this issue, Inspex suggests doing the following actions:

- Set the $LEAF trusted caller as `BigFarm` contract only
- Set the $TREE trusted caller as `GreenHouse` contract only
- Set the `sdToken` trusted caller as `BigFarm` contract only
- Renounce the ownership of all token contracts

5.4. Denial of Services in BigFarm

ID	IDX-004
Target	BigFarm.sol
Category	Advanced Smart Contract Vulnerability
CWE	CWE-755: Improper Handling of Exceptional Conditions
Risk	**Severity: High** **Impact: High** The victim won't be able to execute core functions of the `BigFarm` contract, causing disruption of service and loss of reputation to the platform. **Likelihood: Medium** This attack can be done by anyone to any address without prior deposit; however, there is no direct benefit for the attacker, resulting in low motivation for the attack.
Status	**Resolved** Seeder Finance team has resolved this issue by limiting the minimum farming deposit amount to 1E15 and also allow the beneficiary to withdraw in `BigFarmV3.sol`.

5.4.1. Description

In the `BigFarm` contract, users can deposit tokens specified in each farm to gain $LEAF reward using the `deposit()` function. The `beneficiary` variable in the function can be controlled by the users, allowing the deposit by one address for another address. The first address that deposits for each `beneficiary` will be set as the farm's initiator, preventing others from depositing or withdrawing for that beneficiary.

BigFarm.sol

```
109 function deposit(uint256 farmId, address beneficiary, uint256 amount) external
    {
110     require(_farms.length > 0 && _farms.length.sub(1) >= farmId, "Farm ID out
    of length");
111
112     address inititator = msg.sender;
113     require(inititator != address(0), "Not allow zero address to perform
    deposit");
114
115     Farmer storage farmer = _farmers[farmId][beneficiary];
116     if (farmer.initiator == address(0)) {
117         farmer.initiator = inititator;
118         _updateFarm(farmId);
119     } else {
120         require(farmer.initiator == inititator, "Only initiator can do
```

```
deposit");
        _harvest(farmId, beneficiary);
    }

    Farm memory farm = _farms[farmId];
    farm.stakeToken.transferFrom(inititator, address(this), amount);

    farmer.stakeAmount = farmer.stakeAmount.add(amount);
    farmer.rewardPerStakeWithBuffer = farm.rewardPerStakeWithBuffer;
    emit Deposit(msg.sender, beneficiary, farmId, amount);
}
```

This behavior can be abused by others to disrupt the use of smart contact. Malicious actors can perform a deposit with 0 `amount` for another `beneficiary` without any prior deposit, preventing that `beneficiary` address from being used by the actual owner.

5.4.2. Remediation

Inspex suggests allowing the `beneficiary` to perform withdrawal to return the funds to the initiator, for example:

BigFarm.sol

```
function withdraw(uint256 farmId, address beneficiary, uint256 amount) external
{
    require(_farms.length > 0 && _farms.length.sub(1) >= farmId, "Farm ID out
of length");

    address initiator = msg.sender;
    require(initiator != address(0), "Not allow zero address to perform
withdraw");

    Farmer storage farmer = _farmers[farmId][beneficiary];
    require(farmer.initiator == initiator || initiator == beneficiary, "Only
initiator or beneficiary can withdraw farming");

    _harvest(farmId, beneficiary);

    farmer.stakeAmount = farmer.stakeAmount.sub(amount);
    _farms[farmId].stakeToken.transfer(farmer.initiator, amount);

    if (farmer.stakeAmount == 0) {
        farmer.initiator = address(0);
    }
    emit Withdraw(msg.sender, beneficiary, farmId, amount);
}
```

5.5. Centralized Control of State Variable

ID	IDX-005
Target	BigFarm.sol GranaryV2.sol GreenHouse.sol BankV1_5.sol
Category	General Smart Contract Vulnerability
CWE	CWE-710: Improper Adherence to Coding
Risk	**Severity:** High **Impact:** Medium The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users. **Likelihood:** High There is nothing to restrict the changes from being done; however, the changes are limited by fixed values in the smart contracts.
Status	Resolved Seeder Finance team has resolved this issue by implementing a timelock over the `ProxyAdmin` contract at the following address: 0xc5935ef3d390fde99dc57cc768a674b829e17805 The `Timelock` contract can be found at the following address: 0xbe9E4627881D03940a1E1B64c2D7fFc62813f5a5

5.5.1. Description

Critical state variables can be updated any time by the controlling authorities. Changes in these variables can cause impacts to the users, so the users should accept or be notified before these changes are effective.

However, there is currently no constraint to prevent the authorities from modifying these variables without notifying the users.

The controllable privileged state update functions are as follows:

Target	Function	Modifier
BigFarm.sol (L:176)	addFarm()	onlyOwner
BigFarm.sol (L:193)	setFarm()	onlyOwner
BigFarm.sol (L:202)	setDevAddress()	onlyOwner

BigFarm.sol (L:206)	setPercentLock()	onlyOwner
GranaryV2.sol (L:242)	setKeepPeriod()	onlyOwner
GranaryV2.sol (L:246)	registerGreenHouse()	onlyOwner
GreenHouse.sol (L:58)	setFertilizedTokenAndTreeRatio()	onlyOwner
BankV1_5.sol (L:307)	addDepositPair()	onlyOwner
BankV1_5.sol (L:321)	setDepositFeeRate()	onlyOwner
BankV1_5.sol (L:325)	setWithdrawFeeRate()	onlyOwner
BankV1_5.sol (L:329)	setPlatformAddress()	onlyOwner
BankV1_5.sol (L:333)	setPlatformFeeDividendRate()	onlyOwner

5.5.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. However, if modifications are needed, Inspex suggests limiting the use of these functions via the following options:

- Implementing a community-run governance to control the use of these functions
- Using a `Timelock` contract to delay the changes for a reasonable amount of time

5.6. Improper Share Calculation

ID	IDX-006
Target	BankV1_5.sol
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity: High** **Impact: High** The attacker can drain the tokens from the target contract. **Likelihood: Medium** This vulnerability can easily be exploited; however, the borrowing feature is not yet enabled.
Status	Resolved Seeder Finance team has resolved this issue in `BankV3.sol`.

5.6.1. Description

In the Bank contract, the value of share (`ibToken`) is calculated using the amount of the `originToken` owned by the contract as one of the factors. The current design uses `balanceOf()` function to get the amount of the token within the contract. However, once the borrowing feature is enabled, the calculation of share will be incorrect due to the `totalOrigin` variable in the `deposit()` and `withdraw()` functions, since the borrowed balance is not included.

BankV1_5.sol

```
146 function deposit(address depositOriginToken, uint256 originTokenAmount)
    external {
147     address depositor = msg.sender;
148     require(originTokenAmount >= 1E15, "The amount must be more than or equal
    1E15");
149
150     DepositPair memory depositPair = _tokenDepositPairs[depositOriginToken];
151     require(address(depositPair.originToken) != address(0), "Not support
    token");
152
153     uint256 totalOrigin = depositPair.originToken.balanceOf(address(this));
154     uint256 totalIB = depositPair.ibToken.totalSupply();
155
156     uint256 ibTokenAmount;
157     uint256 originTokenFee;
158     (ibTokenAmount, originTokenFee) = _calculateDeposit(totalOrigin, totalIB,
```

```
    originTokenAmount);
```

This calculation results in an incorrect `ibToken` price, which could be manipulated to drain the funds in the contract by performing the following steps:

- Borrow the target token from the Bank contract to reduce the token balance.
- Deposit the target token to the Bank contract. Due to the lower balance of the target token, an incorrect amount of `ibToken` will be minted.
- Payback the target token to the Bank contract.
- Withdraw the target token from the Bank contract to gain the additional target token.

5.6.2. Remediation

Inspex suggests adding the borrowed amount to the `totalOrigin` variable prior to the share calculation as follows:

BankV1_5.sol

```
146 function deposit(address depositOriginToken, uint256 originTokenAmount)
    external {
147     address depositor = msg.sender;
148     require(originTokenAmount >= 1E15, "The amount must be more than or equal
    1E15");
149
150     DepositPair memory depositPair = _tokenDepositPairs[depositOriginToken];
151     require(address(depositPair.originToken) != address(0), "Not support
    token");
152
153     uint256 totalOrigin = _getTotalTokenBalance(depositPair);
154     uint256 totalIB = depositPair.ibToken.totalSupply();
155
156     uint256 ibTokenAmount;
```

BankV1_5.sol

```
193 function withdraw(address asOriginToken, uint256 ibTokenAmount) external {
194
195     uint256 totalOrigin;
196     uint256 totalIB;
197
198     DepositPair memory depositPair = _tokenDepositPairs[asOriginToken];
199     require(address(depositPair.originToken) != address(0), "Not support
    token");
200     IERC20 originToken = depositPair.originToken;
201     IERC20 ibToken = depositPair.ibToken;
202
203     totalOrigin = _getTotalTokenBalance(depositPair);
```

```
204     totalIB = ibToken.totalSupply();
205
206     uint256 originalTokenAmount;
207     uint256 originTokenFee;
208     (originalTokenAmount, originTokenFee) = _calculateWithdraw(totalOrigin,
    totalIB, ibTokenAmount);
209
210     require(originToken.balanceOf(address(this)) >=
    originalTokenAmount.add(originTokenFee), "Insufficient available balance for
    withdraw due to the borrowing");
211
212     address addr = address(ibToken);
```

Please note that in the `withdraw()` function, the condition in line 210 should be changed to compare the current token balance inside the contract with the balance to be withdrawn.

5.7. Incorrect Token Distribution

ID	IDX-007
Target	BigFarm.sol
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Low The inconsistency between the current contract and document can cause the loss of trust from users. **Likelihood:** High All distributed rewards have been affected with this issue.
Status	Resolved Seeder Finance team has clarified that the 3/20 is out of 160M (user reward) token, which is equal to 24M (dev portion). 24M is equal to 12% of 200M.

5.7.1. Description

For Seeder Finance tokenomics document, please navigate to the following URL:
https://docs.seeder.finance/tokenomics

As mentioned in the document, 12% of $LEAF shall be distributed to the developer address according to the Fair Launch token distribution plan. However, the actual implementation of the token distribution in the `BigFarm` contract does not match with the document.

As shown below, the developer portion's reward amount is calculated by multiplying the total reward amount minted with 3 and divided by 20, resulting in 15% instead of 12%.

BigFarm.sol

```
225 if (totalRewardWithBuffer > 0) {
226     uint256 devPortionWithBuffer = totalRewardWithBuffer.mul(3).div(20);
227     _leaf.mint(devAddress, devPortionWithBuffer.div(CALCULATE_PRECISION));
228     _leaf.mint(address(this), totalRewardWithBuffer.div(CALCULATE_PRECISION));
229 }
```

5.7.2. Remediation

Inspex suggests updating the calculation in line 226 with the following code to be consistent with the document:

BigFarm.sol

```
226     uint256 devPortionWithBuffer = totalRewardWithBuffer.mul(3).div(25);
```

5.8. Improper Reward Calculation (1)

ID	IDX-008
Target	BigFarm.sol
Category	Advanced Smart Contract Vulnerability
CWE	CWE-755: Improper Handling of Exceptional Conditions
Risk	**Severity:** Medium **Impact:** Medium The reward miscalculation can lead to unfair $LEAF token distribution, which may cause loss of reputation. **Likelihood:** Medium This is an edge case that will only happen when the calculation spans over multiple reward slots; however, it affects every pool.
Status	Resolved * Seeder Finance will mitigate this issue by executing `updateAllFarms()` on the block before each new reward period.

5.8.1. Description

In the `BigFarm` contract, the reward minted for the users is calculated using the `_calculatedReward` function. One of the factors used in the calculation is the amount of the total reward minted per block.

BigFarm.sol

```
239 function _calculateReward(Farm memory farm) private view returns (uint256
    additionRewardPerStakeWithBuffer, uint256 totalRewardWithBuffer, uint256
    stakeTokenSupply) {
240     additionRewardPerStakeWithBuffer = 0;
241     totalRewardWithBuffer = 0;
242     stakeTokenSupply = farm.stakeToken.balanceOf(address(this));
243
244     if (stakeTokenSupply > 0 && farm.allocationPoint > 0) {
245         uint256 numberOfBlocks = block.number.sub(farm.lastUpdateBlock);
246         uint256 rewardPerBlockPerAllocation =
    _getRewardPerBlock().mul(farm.allocationPoint).div(totalAllocationPoint);
247         totalRewardWithBuffer =
    numberOfBlocks.mul(rewardPerBlockPerAllocation).mul(CALCULATE_PRECISION);
248         additionRewardPerStakeWithBuffer =
    totalRewardWithBuffer.div(stakeTokenSupply);
249     }
250 }
```

The amount of reward minted per block is determined by the reward plan, and the index of the reward plan is calculated from the current block number.

BigFarm.sol

```
288 function _getRewardPerBlock() private view returns (uint256) {
289     uint256 rewardPerBlock = 0;
290
291     if (_startBlock < block.number) {
292         uint256 currentRewardSlot = _getCurrentRewardSlot();
293         if (currentRewardSlot < _rewardPlan.length) {
294             rewardPerBlock = _rewardPlan[currentRewardSlot];
295         }
296     }
297
298     return rewardPerBlock;
299 }
300
301 function _getCurrentRewardSlot() private view returns (uint256) {
302     uint currentBlock = block.number;
303     if (currentBlock > _startBlock) {
304         uint256 numberOfBlock = currentBlock.sub(_startBlock);
305
306         return numberOfBlock.div(REWARD_BLOCK_WIDE);
307     }
308
309     return 0;
310 }
```

However, there are some edge cases that can cause the reward to be slightly miscalculated. This can happen when the last time the reward is calculated in a different reward slot from the current one.

As an example case, assuming that:

```
_startBlock = 0
REWARD_BLOCK_WIDE = 500
block.number = 1200
farm.lastUpdateBlock = 800
farm.allocationPoint = 10
totalAllocationPoint = 100
_rewardPlan[1] = 60
_rewardPlan[2] = 40
```

If the reward is calculated at the current block, the calculation logic is as follows:

```
block 800 -> block 1200
numberOfBlocks = block.number - farm.lastUpdateBlock = 1200 - 800 = 400
```

```
currentRewardSlot = (block.number - _startBlock) / REWARD_BLOCK_WIDE = 1200 / 500 = 2
rewardPerBlock = _rewardPlan[currentRewardSlot] = _rewardPlan[2] = 40
rewardPerBlockPerAllocation = rewardPerBlock * farm.allocationPoint /
totalAllocationPoint = 40 * 10 / 100 = 4
totalRewardWithBuffer = numberOfBlocks * rewardPerBlockPerAllocation *
CALCULATE_PRECISION = 400 * 4 * 1E18 = 1600 * 1E18
```

This causes the reward during the period between block 800 and block 1000 to be incorrect, since that period should use the value of `_rewardPlan[1]` for the calculation instead of `_rewardPlan[2]`.

5.8.2. Remediation

Inspex suggests fixing the calculation by calculating the sum of the rewards from each reward slot.

5.9. Unsafe Token Transfer

ID	IDX-009
Target	BankV1_5.sol
Category	General Smart Contract Vulnerability
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Medium **Impact:** High `ibToken` can be minted without using any token. **Likelihood:** Low Only improperly implemented token that does not revert the transaction on the invalid transfer amount is affected.
Status	Resolved Seeder Finance team has resolved this issue in `BankV3.sol`.

5.9.1. Description

External ERC20 tokens can be added to the contract as an `originToken` using the `addDepositPair()` function. ERC20 tokens can be improperly implemented, allowing the execution of failed `transfer()` and `transferFrom()` functions without reverting when the invalid transfer amount occurs. This can cause significant damage to the smart contract if not enough token is available.

For example, in the `deposit()` function, if the user sets high `originTokenAmount` with an insufficient amount of `depositPair.originToken`, but there is no reverting in the `transferFrom()` and `transfer` function, the `deposit()` transaction can be done successfully. This can cause `ibToken` to be minted without having any token transferred to the contract.

BankV1_5.sol

```
146 function deposit(address depositOriginToken, uint256 originTokenAmount)
    external {
147     address depositor = msg.sender;
148     require(originTokenAmount >= 1E15, "The amount must be more than or equal
    1E15");
149
150     DepositPair memory depositPair = _tokenDepositPairs[depositOriginToken];
151     require(address(depositPair.originToken) != address(0), "Not support
    token");
152
153     uint256 totalOrigin = depositPair.originToken.balanceOf(address(this));
154     uint256 totalIB = depositPair.ibToken.totalSupply();
```

```
155
156     uint256 ibTokenAmount;
157     uint256 originTokenFee;
158     (ibTokenAmount, originTokenFee) = _calculateDeposit(totalOrigin, totalIB,
    originTokenAmount);
159     uint256 originTokenPlatformFee =
    originTokenFee.mul(_platformFeeMultiplyer).div(_platformFeeDivider);
160
161     depositPair.originToken.transferFrom(depositor, address(this),
    originTokenAmount);
162     depositPair.originToken.transfer(platformAddress, originTokenPlatformFee);
163
164     (bool success, bytes memory result) =
    address(depositPair.ibToken).call(abi.encodeWithSignature("mint(address,uint256
    )", depositor, ibTokenAmount));
165     require(success, string(result));
166
167     emit Deposit(depositor, address(depositPair.originToken),
    address(depositPair.ibToken), originTokenAmount, ibTokenAmount);
168 }
```

The affected tokens are as follows:

Target	Variable
BigFarm.sol	`stakeToken` of `Farm`
BankV1_5.sol	`originToken` of `DepositPair`

5.9.2. Remediation

Inspex suggests replacing the `transfer()` and `transferFrom()` functions of the untrusted tokens with `safeTransfer()` and `safeTransferFrom()` functions from OpenZeppelin's `SafeERC20` contract, for example:

BankV1_5.sol

```
1  // SPDX-License-Identifier: GPL-3.0
2  pragma solidity ^0.8.0;
3
4  import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
5  import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
6  import "hardhat/console.sol";
7  import "../utils/SafeMath.sol";
8  import "../utils/Loanable.sol";
9  import "../wallet/PlatformWallet.sol";
10
11
12 contract BankV1_5 is Loanable {
13     using SafeMath for uint256;
14     using SafeERC20 for IERC20;
```

BankV1_5.sol

```
161     depositPair.originToken.safeTransferFrom(depositor, address(this),
originTokenAmount);
162     depositPair.originToken.safeTransfer(platformAddress,
originTokenPlatformFee);
```

5.10. Inconsistent Token Transfer Fee

ID	IDX-010
Target	Tree.sol
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Low Users can use the `transferFrom()` function to avoid the transfer fee. **Likelihood:** High The fee will always be inconsistent between `transfer()` and `transferFrom()` functions.
Status	Resolved * Seeder Finance team has decided to remove the transfer fee by setting the fee to 0 and set the owner of the contract as the `Timelock` contract.

5.10.1. Description

The $TREE has a transfer fee implemented in the override implementation of `transfer()` function. However, there is no fee collected in the `transferFrom()` function.

Tree.sol

```
39  function transfer(address recipient, uint256 amount) public override returns
    (bool) {
40      uint256 feeAmount =
    amount.mul(transferFeeMultiplyer).div(transferFeeDivider);
41      uint256 amountExcludeFee = amount.sub(feeAmount);
42
43      _transfer(_msgSender(), platformAddress, feeAmount);
44      _transfer(_msgSender(), recipient, amountExcludeFee);
45
46      return true;
47  }
```

This can cause inconsistencies in other smart contracts if the difference in fee is not considered. Also, the users can avoid the transfer fee by using the `transferFrom()` function.

5.10.2. Remediation

Inspex suggests adding the fee to the `transferFrom()` function to keep the fee consistent, for example:

Tree.sol

```
39  function transfer(address recipient, uint256 amount) public override returns
    (bool) {
40      uint256 feeAmount =
    amount.mul(transferFeeMultiplyer).div(transferFeeDivider);
41      uint256 amountExcludeFee = amount.sub(feeAmount);
42
43      _transfer(_msgSender(), platformAddress, feeAmount);
44      _transfer(_msgSender(), recipient, amountExcludeFee);
45
46      return true;
47  }
48
49  function transferFrom(address sender, address recipient, uint256 amount) public
    virtual override returns (bool) {
50      uint256 feeAmount =
    amount.mul(transferFeeMultiplyer).div(transferFeeDivider);
51      uint256 amountExcludeFee = amount.sub(feeAmount);
52      _transfer(sender, platformAddress, feeAmount);
53      _transfer(sender, recipient, amountExcludeFee);
54
55      uint256 currentAllowance = _allowances[sender][_msgSender()];
56      require(currentAllowance >= amount, "ERC20: transfer amount exceeds
    allowance");
57      unchecked {
58          _approve(sender, _msgSender(), currentAllowance - amount);
59      }
60
61      return true;
62  }
```

5.11. Design Flaw in updateAllFarms() Function

ID	IDX-011
Target	BigFarm.sol
Category	General Smart Contract Vulnerability
CWE	CWE-400: Uncontrolled Resource Consumption
Risk	**Severity:** Low **Impact:** Medium The `updateAllFarms()` function will eventually be unusable due to excessive gas usage. **Likelihood:** Low It is very unlikely that the `_farms` size will be raised until the `updateAllFarms()` is eventually unusable.
Status	**Acknowledged** Seeder Finance team has acknowledged this issue. The team has prepared a UAT environment which has the same number of farms as mainnet, so this problem can be proactively prevented.

5.11.1. Description

The `updateAllFarms()` function executes the `_updateFarm()` function, which is a state modifying function for all added farms as shown below:

BigFarm.sol

```
161 function updateAllFarms() public {
162     for (uint256 farmId = 0; farmId < _farms.length; farmId++) {
163         _updateFarm(farmId);
164     }
165 }
```

With the current design, the added farms cannot be removed. They can only be disabled by setting the `farm.allocationPoint` to 0. Even if a farm is disabled, the `_updateFarm()` function for this farm is still called. Therefore, if new farms continue to be added to this contract, the `_farms.length` will continue to grow and this function will eventually be unusable due to excessive gas usage.

5.11.2. Remediation

Inspex suggests making the contract capable of removing unnecessary or ended farms to reduce the loop round in the `updateAllFarms()` function, for example:

```
require(farmId < _farms.length);
_farms[farmId] = _farms[_farms.length-1];
_farms.length--;
```

5.12. Improper Reward Calculation (2)

ID	IDX-012
Target	BigFarm.sol
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Low **Impact:** Low The reward of the pool that has the same staking token as the reward token will be slightly lower than what it should be. **Likelihood:** Medium It is unlikely that there is a pool that has the same staking token as the reward token.
Status	Resolved Seeder Finance team has resolved this issue in `BigFarmV3.sol`.

5.12.1. Description

In the `BigFarm` contract, a new staking pool can be added using the `addFarm()` function. The staking token for the new pool is defined using the `stakeToken` variable; however, there is no additional checking whether the `stakeToken` is the same as the reward token (`_leaf`) or not.

BigFarm.sol

```
176 function addFarm(address stakeToken, uint256 allocationPoint, uint256
    startBlock) external onlyOwner {
177     require(!_doesFarmExist(stakeToken), "This Farm already exist");
178
179     updateAllFarms();
180     uint256 lastUpdateBlock = block.number > startBlock ? block.number :
    startBlock;
181     totalAllocationPoint = totalAllocationPoint.add(allocationPoint);
182
183     Farm memory farm = Farm({
184         stakeToken: IERC20(stakeToken),
185         allocationPoint: allocationPoint,
186         lastUpdateBlock: lastUpdateBlock,
187         rewardPerStakeWithBuffer: 0
188     });
189
190     _farms.push(farm);
191 }
```

When the `stakeToken` is the same token as `_leaf`, reward calculation for that pool in the `_calculateReward()` function can be incorrect. This is because the current balance of the `stakeToken` in the contract is used in the calculation of the reward. Since the `stakeToken` is the same token as the reward, the reward minted to the contract will inflate the value of `stakeTokenSupply`, causing the reward of that pool to be less than what it should be.

BigFarm.sol

```
239 function _calculateReward(Farm memory farm) private view returns (uint256
    additionRewardPerStakeWithBuffer, uint256 totalRewardWithBuffer, uint256
    stakeTokenSupply) {
240     additionRewardPerStakeWithBuffer = 0;
241     totalRewardWithBuffer = 0;
242     stakeTokenSupply = farm.stakeToken.balanceOf(address(this));
243
244     if (stakeTokenSupply > 0 && farm.allocationPoint > 0) {
245             uint256 numberOfBlocks = block.number.sub(farm.lastUpdateBlock);
246             uint256 rewardPerBlockPerAllocation =
    _getRewardPerBlock().mul(farm.allocationPoint).div(totalAllocationPoint);
247             totalRewardWithBuffer =
    numberOfBlocks.mul(rewardPerBlockPerAllocation).mul(CALCULATE_PRECISION);
248             additionRewardPerStakeWithBuffer =
    totalRewardWithBuffer.div(stakeTokenSupply);
249     }
250 }
```

5.12.2. Remediation

Inspex suggests checking the value of the `stakeToken` in the `addFarm()` function to prevent the pool with the same staking token as the reward token from being added , for example:

BigFarm.sol

```
176 function addFarm(address stakeToken, uint256 allocationPoint, uint256
    startBlock) external onlyOwner {
177     require(!_doesFarmExist(stakeToken), "This Farm already exist");
178     require(stakeToken != address(_leaf)), "Stake token is same as reward");
179
180     updateAllFarms();
181     uint256 lastUpdateBlock = block.number > startBlock ? block.number :
    startBlock;
182     totalAllocationPoint = totalAllocationPoint.add(allocationPoint);
183
184     Farm memory farm = Farm({
185         stakeToken: IERC20(stakeToken),
186         allocationPoint: allocationPoint,
187         lastUpdateBlock: lastUpdateBlock,
188         rewardPerStakeWithBuffer: 0
```

```
189     });
190
191     _farms.push(farm);
192 }
```

However, if the pool with the same staking token as the reward token is required, Inspex suggests minting the reward token to another contract to prevent the amount of the staked token from being mixed up with the reward token.

5.13. Improper Release of Locked Tokens

ID	IDX-013
Target	GranaryV2.sol
Category	Advanced Smart Contract Vulnerability
CWE	CWE-755: Improper Handling of Exceptional Conditions
Risk	**Severity:** Low **Impact:** Low The tokens to be released in the later indexes won't be released until all of the prior entries are done, and may cause loss of reputation to the platform. **Likelihood:** Medium This issue will happen when the keepPeriodInSecond is set to a lower value; however, it is unlikely for this value to be frequently changed.
Status	Resolved Seeder Finance team has resolved this issue by removing the `setKeepPeriod()` function in `GranaryV3.sol`.

5.13.1. Description

In the `GranaryV2` contract, `keepPeriodInSecond` can be updated using the `setKeepPeriod()` function.

GranaryV2.sol

```
242 function setKeepPeriod(uint256 numberOfDay) external onlyOwner {
243     keepPeriodInSecond = numberOfDay.mul(SECOND_PER_DAY);
244 }
```

The `keepPeriodInSecond` variable is used to determine the release times in the `keep()` function. The release times are pushed into `releaseTimes1` and `releaseTimes2` arrays.

GranaryV2.sol

```
114 function keep(address beneficiary, uint256 amount) external {
115     require(beneficiary != address(0), "Cannot keep record for zero address");
116
117     keepToken.transferFrom(msg.sender, address(this), amount);
118
119     uint256 releaseTime1 =
    block.timestamp.add(keepPeriodInSecond).div(SECOND_PER_DAY);
120     uint256 releaseTime2 =
    block.timestamp.add(keepPeriodInSecond.mul(2)).div(SECOND_PER_DAY);
121
```

```
122     uint256 amount1 = amount.div(2);
123     uint256 amount2 = amount.sub(amount1);
124
125     // Add release time and amount for 1st keep record
126     uint256 processingIndex1 = processingIndexes1[beneficiary];
127     uint256[] storage ownerReleaseTimes1 = releaseTimes1[beneficiary];
128     if (ownerReleaseTimes1.length == 0 ||
129             processingIndex1 >= ownerReleaseTimes1.length ||
130             ownerReleaseTimes1[ownerReleaseTimes1.length - 1] != releaseTime1)
    {
131         releaseTimes1[beneficiary].push(releaseTime1);
132     }
133     releaseAmounts1[beneficiary][releaseTime1] =
    releaseAmounts1[beneficiary][releaseTime1].add(amount1);
134
135     // Add release time and amount for 2nd keep record
136     uint256 processingIndex2 = processingIndexes2[beneficiary];
137     uint256[] storage ownerReleaseTimes2 = releaseTimes2[beneficiary];
138     if (ownerReleaseTimes2.length == 0 ||
139             processingIndex2 >= ownerReleaseTimes2.length ||
140             ownerReleaseTimes2[ownerReleaseTimes2.length - 1] != releaseTime2)
    {
141         releaseTimes2[beneficiary].push(releaseTime2);
142     }
143     releaseAmounts2[beneficiary][releaseTime2] =
    releaseAmounts2[beneficiary][releaseTime2].add(amount2);
144
145     emit Keep(beneficiary, amount);
146 }
```

However, if the `keepPeriodInSecond` is updated to a lower value, the release times of newer entries can be lower than the entries in the lower indexes. With the current logic of `release()` function, the balances stored in the higher indexes can't be released until all of the lower indexes are released. This will happen even when the current time has reached the release time of the entries in the higher indexes due to the break in the else condition.

GranaryV2.sol

```
148 function release(uint256[] calldata releaseV1Items) external {
149     address beneficiary = msg.sender;
150     uint256 releaseBalance = 0;
151
152     // V1
153     for (uint256 index = 0; index < releaseV1Items.length; index++) {
154         uint256 itemIndex = releaseV1Items[index];
155         Record storage record = keepRecords[beneficiary][itemIndex];
156
```

```
157         require(block.timestamp >= record.releaseTimestamp, "Granary: Release
too early");
158         releaseBalance = releaseBalance.add(record.amount);
159 
160         delete keepRecords[beneficiary][itemIndex];
161     }
162 
163     // V2
164     uint256 currentTime = block.timestamp.div(SECOND_PER_DAY);
165     for (uint256 index = processingIndexes1[beneficiary]; index <
releaseTimes1[beneficiary].length; index++) {
166         uint256 releaseTime = releaseTimes1[beneficiary][index];
167         if (currentTime >= releaseTime) {
168             releaseBalance =
releaseBalance.add(releaseAmounts1[beneficiary][releaseTime]);
169             processingIndexes1[beneficiary] = processingIndexes1[beneficiary] +
1;
170             delete releaseAmounts1[beneficiary][releaseTime];
171         } else {
172             break;
173         }
174     }
175 
176     for (uint256 index = processingIndexes2[beneficiary]; index <
releaseTimes2[beneficiary].length; index++) {
177         uint256 releaseTime = releaseTimes2[beneficiary][index];
178         if (currentTime >= releaseTime) {
179             releaseBalance =
releaseBalance.add(releaseAmounts2[beneficiary][releaseTime]);
180             processingIndexes2[beneficiary] = processingIndexes2[beneficiary] +
1;
181             delete releaseAmounts2[beneficiary][releaseTime];
182         } else {
183             break;
184         }
185     }
186 
187     // Release balance
188     require(releaseBalance > 0, "GranaryV2: No balance to be released");
189     keepToken.transfer(beneficiary, releaseBalance);
190     emit Release(beneficiary, releaseBalance);
191 }
```

5.13.2. Remediation

Inspex suggests removing the else condition to check the release time of every unprocessed indexes, for example:

GranaryV2.sol

```
148 function release(uint256[] calldata releaseV1Items) external {
149     address beneficiary = msg.sender;
150     uint256 releaseBalance = 0;
151 
152     // V1
153     for (uint256 index = 0; index < releaseV1Items.length; index++) {
154         uint256 itemIndex = releaseV1Items[index];
155         Record storage record = keepRecords[beneficiary][itemIndex];
156 
157         require(block.timestamp >= record.releaseTimestamp, "Granary: Release
    too early");
158         releaseBalance = releaseBalance.add(record.amount);
159 
160         delete keepRecords[beneficiary][itemIndex];
161     }
162 
163     // V2
164     uint256 currentTime = block.timestamp.div(SECOND_PER_DAY);
165     for (uint256 index = processingIndexes1[beneficiary]; index <
    releaseTimes1[beneficiary].length; index++) {
166         uint256 releaseTime = releaseTimes1[beneficiary][index];
167         if (currentTime >= releaseTime) {
168             releaseBalance =
    releaseBalance.add(releaseAmounts1[beneficiary][releaseTime]);
169             processingIndexes1[beneficiary] = processingIndexes1[beneficiary] +
    1;
170             delete releaseAmounts1[beneficiary][releaseTime];
171         }
172     }
173 
174     for (uint256 index = processingIndexes2[beneficiary]; index <
    releaseTimes2[beneficiary].length; index++) {
175         uint256 releaseTime = releaseTimes2[beneficiary][index];
176         if (currentTime >= releaseTime) {
177             releaseBalance =
    releaseBalance.add(releaseAmounts2[beneficiary][releaseTime]);
178             processingIndexes2[beneficiary] = processingIndexes2[beneficiary] +
    1;
179             delete releaseAmounts2[beneficiary][releaseTime];
180         }
181     }
```

```
182
183     // Release balance
184     require(releaseBalance > 0, "GranaryV2: No balance to be released");
185     keepToken.transfer(beneficiary, releaseBalance);
186     emit Release(beneficiary, releaseBalance);
187 }
```

5.14. Inexplicit Solidity Compiler Version

ID	IDX-014
Target	BigFarm.sol GranaryV2.sol GreenHouse.sol BankV1_5.sol Leaf.sol Tree.sol sdToken.sol
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact:** None **Likelihood:** None
Status	Resolved Seeder Finance team has resolved this issue by fixing the version number to 0.8.4.

5.14.1. Description

The Solidity compiler versions declared in the smart contracts were not explicit. Each compilation may be done using different compiler versions, which may potentially result in compatibility issues.

BigFarm.sol

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity ^0.8.0;
```

5.14.2. Remediation

Inspex suggests fixing the solidity compiler to the latest stable version. As of June 2021, the latest stable versions of Solidity compiler in major 0.8 is v0.8.5

BigFarm.sol

```
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity 0.8.5;
```

5.15. Improper Function Visibility

ID	IDX-015
Target	GranaryV2.sol
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact:** None **Likelihood:** None
Status	Resolved Seeder Finance team has resolved this issue in `GranaryV3.sol`.

5.15.1. Description

The `getKeepInformations()` function in `GranaryV2` has public visibility and is never called by any internal function.

Functions with public visibility copy calldata to memory when being executed, while external functions can read directly from calldata. Memory allocation uses more resources (gas) than reading directly from calldata.

The following source code shows that the `getKeepInformations()` function of the `GranaryV2` contract is set to public and it is never called from any internal function.

GranaryV2.sol

```
62 function getKeepInformations(address beneficiary) public view returns
   (uint256[] memory v1ReleaseTimes, uint256[] memory v1ReleaseAmounts,

63 uint256[] memory v2ReleaseTimes, uint256[] memory v2ReleaseAmounts) {
```

5.15.2. Remediation

Inspex suggests changing the `getKeepInformations()` function visibility to `external` if it is not called by any internal function as shown in the following example:

GranaryV2.sol

```
62 function getKeepInformations(address beneficiary) external view returns
   (uint256[] memory v1ReleaseTimes, uint256[] memory v1ReleaseAmounts,

63 uint256[] memory v2ReleaseTimes, uint256[] memory v2ReleaseAmounts) {
```

5.16. Design Flaw in the Loanable Library

ID	IDX-016
Target	Loanable.sol
Category	Advanced Smart Contract Vulnerability
CWE	CWE-755: Improper Handling of Exceptional Conditions
Risk	**Severity:** Info **Impact:** None **Likelihood:** None
Status	Resolved Seeder Finance team has resolved this issue in `LoanableV3.sol`.

5.16.1. Description

In the `Loanable` contract, a loan issuer can be added using the `addIssuer()` function. The function then calls the `_addLoanIssuer()` function.

Loanable.sol

```
27 function addIssuer(address loanIssuerAddress) external onlyOwner {
28     _addLoanIssuer(loanIssuerAddress);
29 }
```

However, there is no checking done on the value of `loanIssuerAddress` parameter, so duplicated loan issuers can be added.

Loanable.sol

```
39 function _addLoanIssuer(address loanIssuerAddress) private {
40     _loanIssuers.push(loanIssuerAddress);
41     _loanIssuerMap[loanIssuerAddress] = _loanIssuers.length - 1;
42 }
```

When a duplicated loan issuer is added, the original `_loanIssuerMap` mapping of that address is replaced, so that loan issue can't be removed from the `_loanIssuers` array.

Loanable.sol

```
44 function _removeLoanIssuer(address loanIssuerAddress) private {
45     uint256 index = _loanIssuerMap[loanIssuerAddress];
46     _loanIssuerMap[loanIssuerAddress] = 0;
47     delete _loanIssuers[index];
```

```
48 }
```

Since the functions directly related to the `Loanable` are not currently in use, there is no security impact.

5.16.2. Remediation

Inspex suggests checking the existence of the loan issuer before adding a new one with the same address, for example:

```
39 function _addLoanIssuer(address loanIssuerAddress) private {
40     require(_loanIssuerMap[loanIssuerAddress] == 0, "loanIssuerAddress is
   already exists")
41     _loanIssuers.push(loanIssuerAddress);
42     _loanIssuerMap[loanIssuerAddress] = _loanIssuers.length - 1;
43 }
```

6. Appendix

6.1. About Inspex

Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

Website	https://inspex.co
Twitter	@InspexCo
Facebook	https://www.facebook.com/InspexCo
Telegram	@inspex_announcement

6.2. References

[1] “OWASP Risk Rating Methodology.” [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology. [Accessed: 08-May-2021]

inspeX

CYBERSECURITY PROFESSIONAL SERVICE