HALBORN

Beanstalk - Sunrise Improvements

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **November 13th, 2022 - January 13th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	11/23/2022	Kaan Caglan
0.2	Draft Review	12/05/2022	Roberto Reigada
0.3	Draft Review	12/07/2022	Gabi Urrutia
1.0	Remediation Plan	12/07/2022	Kaan Caglan
1.1	Remediation Plan Review	12/08/2022	Roberto Reigada
1.2	Remediation Plan Review	12/08/2022	Piotr Cielas
1.3	Remediation Plan Review	12/08/2022	Gabi Urrutia
2.0	Scope Update	01/13/2023	Roberto Reigada
2.1	Scope Update Review	01/13/2023	Piotr Cielas
2.2	Scope Update Review	01/13/2023	Gabi Urrutia
2.3	Scope Update	02/24/2023	Roberto Reigada
2.4	Scope Update Review	02/27/2023	Ataberk Yavuzer
2.5	Scope Update Review	02/28/2023	Piotr Cielas
2.6	Scope Update Review	02/28/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com
Roberto Reigada	Halborn	Roberto.Reigada@halborn.com
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Beanstalk is a permissionless fiat stablecoin protocol built on Ethereum that uses credit instead of collateral to issue its native stablecoin.

Beanstalk engaged Halborn to conduct a security audit on their smart contracts beginning on November 13th, 2022 and ending on December 5th, 2022. The security assessment was scoped to the smart contracts provided in the GitHub repository BeanstalkFarms/Beanstalk/tree/bip-30.

New code was introduced by BeanStalk and audited by Halborn Jan 5th 2023. No security issues were identified.

1.2 AUDIT SUMMARY

The team at Halborn was provided 2 weeks for the initial engagement and assigned a full-time security engineer to audit the security of the smart contract. An extra week was taken to audit the new updated code. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing and smart-contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

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

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

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard

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

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

RISK METHODOLOGY:

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

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.

2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

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

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

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

- Oracle.sol
- SeasonFacet.sol
- Sun.sol
- Weather.sol
- FieldFacet.sol
- FundraiserFacet.sol
- LibCurveOracle.sol
- LibTransfer.sol
- LibDibbler.sol
- LibIncentive.sol
- LibPRBMath.sol

Initial commit ID:
- 83469305e606013980cf285f115f550d0586718e

These were all the code changes done between 6699e071626a17283facc67242536037989ecd91 and 83469305e606013980cf285f115f550d0586718e

Fixed commit ID:
- 1454d41f6b9e9b1f45e7662c67e90c31c694c8ea

New code was introduced by BeanStalk and audited by Halborn 5th of January, 2023:
- d67bbdb2319710270d73f14b343738eb60537460. The main changes include:
- Converting the if ladder on the morning auction to a binary search for gas efficiency
- Fixing an incorrect change from the weather struct in appStorage.sol
- Various semantic and formatting changes.

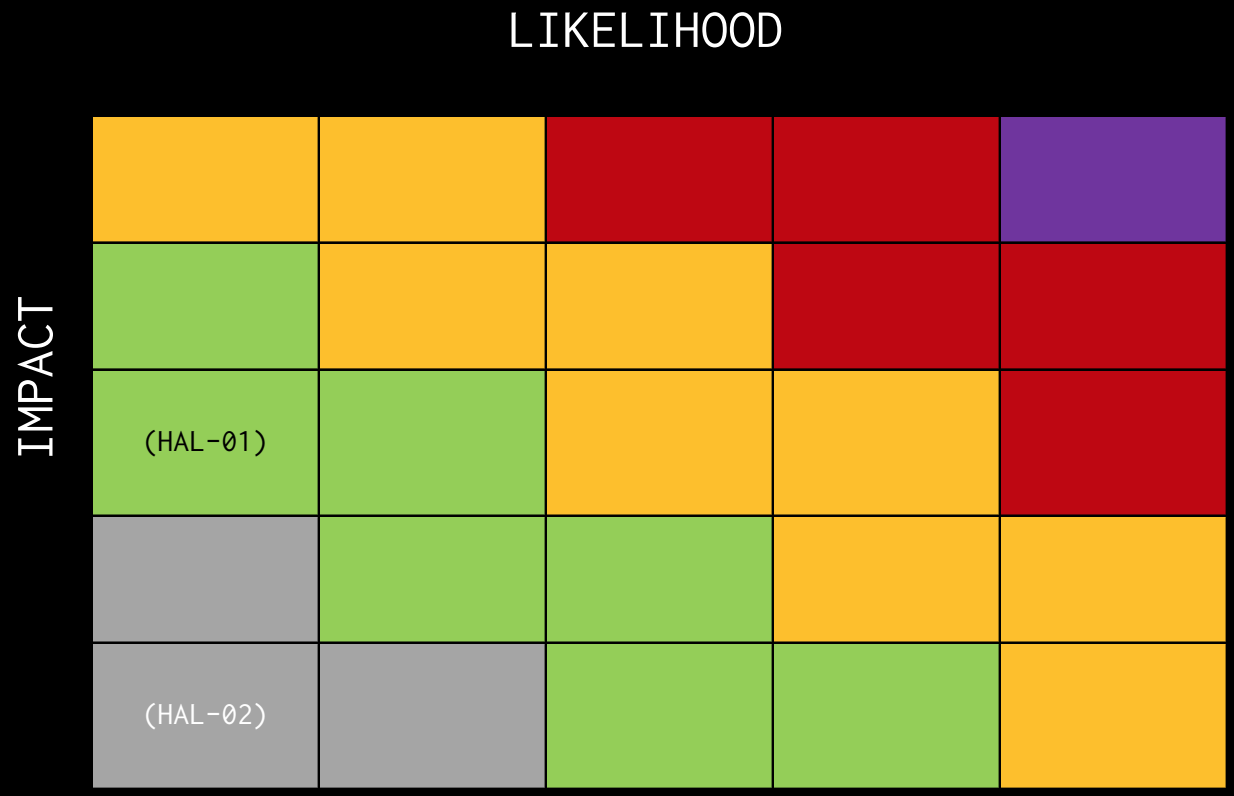
No issues were found in d67bbdb2319710270d73f14b343738eb60537460.

Again, new code was introduced by BeanStalk and audited by Halborn 21st of February, 2023:
- f37cb42809fb8dfc9a0f2891db1ad96a1b848a4c. The main changes are included here.

No issues were found in f37cb42809fb8dfc9a0f2891db1ad96a1b848a4c.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	1



EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - MISSING CONTROL IN MORNINGAUCTION FUNCTION	Low	SOLVED - 12/07/2022
HAL02 - UNUSED STATE VARIABLES	Informational	SOLVED - 12/07/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) MISSING CONTROL IN THE MORNINGAUCTION FUNCTION - LOW

Description:

In the LibDibbler.sol library, there is a function named morningAction(). That function is responsible for returning the temperature value scaled down through a Dutch Auction. The temperature increases over the course of the first 5 minutes (25 blocks) of the season (up to the current temperature). When uint256 delta = block.number.sub(s.season.sunriseBlock); is calculated, the higher the delta the higher the value returned by the function. That function is used in the beansToPodsAbovePeg function later to increase the pods.
However, if s.w.yield is 0 and delta is less than 24 the yield will unintentionally be 1% higher.

Code Location:

Listing 1: LibDibbler.sol (Lines 89,90,147)

```
83      /// @dev function returns the weather scaled down
84      /// @notice based on the block delta
85      // precision level 1e6, as soil has 1e6 precision (1% = 1e6)
86      function morningAuction() internal view returns (uint256) {
87          AppStorage storage s = LibAppStorage.diamondStorage();
88          uint256 delta = block.number.sub(s.season.sunriseBlock);
89          if (delta > 24) { // check most likely case first
90              return uint256(s.w.yield).mul(DECIMALS);
91          } else if (delta == 1) {
92              return auctionMath(279415312704);
93          } else if (delta == 2) {
94              return auctionMath(409336034395);
95          } else if (delta == 3) {
96              return auctionMath(494912626048);
97          } else if (delta == 4) {
98              return auctionMath(558830625409);
99          } else if (delta == 5) {
100             return auctionMath(609868162219);
101         } else if (delta == 6) {
```

```
102             return auctionMath(652355825780);
103         } else if (delta == 7) {
104             return auctionMath(688751347100);
105         } else if (delta == 8) {
106             return auctionMath(720584687295);
107         } else if (delta == 9) {
108             return auctionMath(748873234524);
109         } else if (delta == 10) {
110             return auctionMath(774327938752);
111         } else if (delta == 11) {
112             return auctionMath(797465225780);
113         } else if (delta == 12) {
114             return auctionMath(818672068791);
115         } else if (delta == 13) {
116             return auctionMath(838245938114);
117         } else if (delta == 14) {
118             return auctionMath(856420437864);
119         } else if (delta == 15) {
120             return auctionMath(873382373802);
121         } else if (delta == 16) {
122             return auctionMath(889283474924);
123         } else if (delta == 17) {
124             return auctionMath(904248660443);
125         } else if (delta == 18) {
126             return auctionMath(918382006208);
127         } else if (delta == 19) {
128             return auctionMath(931771138485);
129         } else if (delta == 20) {
130             return auctionMath(944490527707);
131         } else if (delta == 21) {
132             return auctionMath(956603996980);
133         } else if (delta == 22) {
134             return auctionMath(968166659804);
135         } else if (delta == 23) {
136             return auctionMath(979226436102);
137         } else if (delta == 24) {
138             return auctionMath(989825252096);
139         } else {
140             return DECIMALS; //minimium 1% yield
141         }
142     }
143
144     /// @dev scales down weather, minimum 1e6
```

```
145     function auctionMath(uint256 a) private view returns (uint256)
 ↳  {
146         AppStorage storage s = LibAppStorage.diamondStorage();
147         return uint256(s.w.yield).mulDiv(a,1e6).max(DECIMALS);
148     }
```

Proof of Concept:

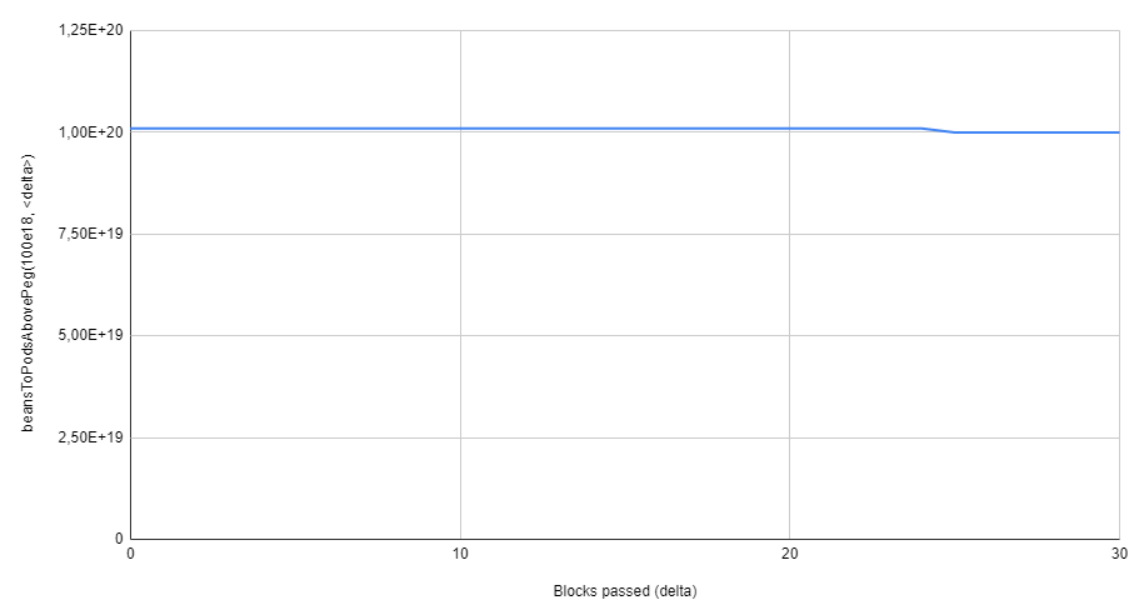
With:
s.w.yield = 0
beans = 100e18

Listing 2

```
 1 contract_Test2.morningAuction(0, 0) -> 1000000
 2 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
 ↳  -> 10100000000000000000
 3 contract_Test2.morningAuction(0, 1) -> 1000000
 4 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
 ↳  -> 10100000000000000000
 5 contract_Test2.morningAuction(0, 2) -> 1000000
 6 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
 ↳  -> 10100000000000000000
 7 contract_Test2.morningAuction(0, 3) -> 1000000
 8 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
 ↳  -> 10100000000000000000
 9 contract_Test2.morningAuction(0, 4) -> 1000000
10 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
 ↳  -> 10100000000000000000
11 contract_Test2.morningAuction(0, 5) -> 1000000
12 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
 ↳  -> 10100000000000000000
13 contract_Test2.morningAuction(0, 6) -> 1000000
14 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
 ↳  -> 10100000000000000000
15 contract_Test2.morningAuction(0, 7) -> 1000000
16 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
 ↳  -> 10100000000000000000
17 contract_Test2.morningAuction(0, 8) -> 1000000
18 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
 ↳  -> 10100000000000000000
19 contract_Test2.morningAuction(0, 9) -> 1000000
```

```
20 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
21 contract_Test2.morningAuction(0, 10) -> 1000000
22 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
23 contract_Test2.morningAuction(0, 11) -> 1000000
24 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
25 contract_Test2.morningAuction(0, 12) -> 1000000
26 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
27 contract_Test2.morningAuction(0, 13) -> 1000000
28 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
29 contract_Test2.morningAuction(0, 14) -> 1000000
30 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
31 contract_Test2.morningAuction(0, 15) -> 1000000
32 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
33 contract_Test2.morningAuction(0, 16) -> 1000000
34 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
35 contract_Test2.morningAuction(0, 17) -> 1000000
36 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
37 contract_Test2.morningAuction(0, 18) -> 1000000
38 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
39 contract_Test2.morningAuction(0, 19) -> 1000000
40 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
41 contract_Test2.morningAuction(0, 20) -> 1000000
42 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
43 contract_Test2.morningAuction(0, 21) -> 1000000
44 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
45 contract_Test2.morningAuction(0, 22) -> 1000000
46 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
47 contract_Test2.morningAuction(0, 23) -> 1000000
48 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
```

```
49 contract_Test2.morningAuction(0, 24) -> 1000000
50 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 1000000)
↳  -> 10100000000000000000
51 contract_Test2.morningAuction(0, 25) -> 0
52 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 0) ->
↳ 10000000000000000000
53 contract_Test2.morningAuction(0, 26) -> 0
54 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 0) ->
↳ 10000000000000000000
55 contract_Test2.morningAuction(0, 27) -> 0
56 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 0) ->
↳ 10000000000000000000
57 contract_Test2.morningAuction(0, 28) -> 0
58 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 0) ->
↳ 10000000000000000000
59 contract_Test2.morningAuction(0, 29) -> 0
60 contract_Test2.beansToPodsAbovePeg(10000000000000000000, 0) ->
↳ 10000000000000000000
```



Recommendation:

It is recommended to add a special case to handle when `s.w.yield` is `0` and `delta > 24`.

Remediation Plan:

SOLVED: The Beanstalk team fixed the issue by adding a 0 control to s.w.field variable in auctionMath function.

Listing 3: LibDibbler.sol (Line 150)

```
146     /// @dev scales down temperature, minimum 1e6 (unless
 ↳ temperature is 0%)
147     function auctionMath(uint256 a) private view returns (uint256)
 ↳  {
148         AppStorage storage s = LibAppStorage.diamondStorage();
149         uint256 _yield  = s.w.yield;
150         if(_yield == 0) return 0;
```

Fixed Commit ID: 1454d41f6b9e9b1f45e7662c67e90c31c694c8ea

3.2 (HAL-02) UNUSED STATE VARIABLES - INFORMATIONAL

Description:

The DECIMAL state variable within the FieldFacet.sol contract is not used. Similarly, the situation also occurs in the LibDibbler library with the constants:
- MAX_BLOCK_ELAPSED
- DENOMINATOR
- SCALE

Code Location:

Listing 4: FieldFacet.sol (Line 23)

```
23         uint128 private constant DECIMAL = 1e6;
24     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

If a state variable will not be used, it is recommended to remove it to save gas.

Remediation Plan:

SOLVED: The Beanstalk team solved the issue.

Fixed Commit ID: 1454d41f6b9e9b1f45e7662c67e90c31c694c8ea

MANUAL TESTING

4.1 Description

Halborn performed manual tests on all the different facets of **BeanStalk**, looking for business logic flaws and security vulnerabilities.

During those manual tests, the following areas were carefully reviewed:

1. Sunrise Rewards

4.2 Results

Beanstalk was overpaying for the `sunrise()` function calls, so it was proposed to change the base incentive for Farmers to call the `sunrise()` function. The solution implemented seeks to minimize the number of Beans that Beanstalk needs to pay for the `sunrise()` call and computes an on-chain estimation of the below formula while adding sufficient tunable parameters to properly account for estimation error and potential manipulation.

The cost to execute a `sunrise()` transaction in Beans is:
`gasUsed * (baseGasFee + priorityFee)* beanEthPrice`

Overall, Beanstalk would strictly mint less than or equal the current Beans that it does currently. Also, the exponential increase in Beans paid for calling the `sunrise()` function late should remain, switching to `blockNumber` instead of timestamp.

An example of the rewards given can be seen below:

Listing 5

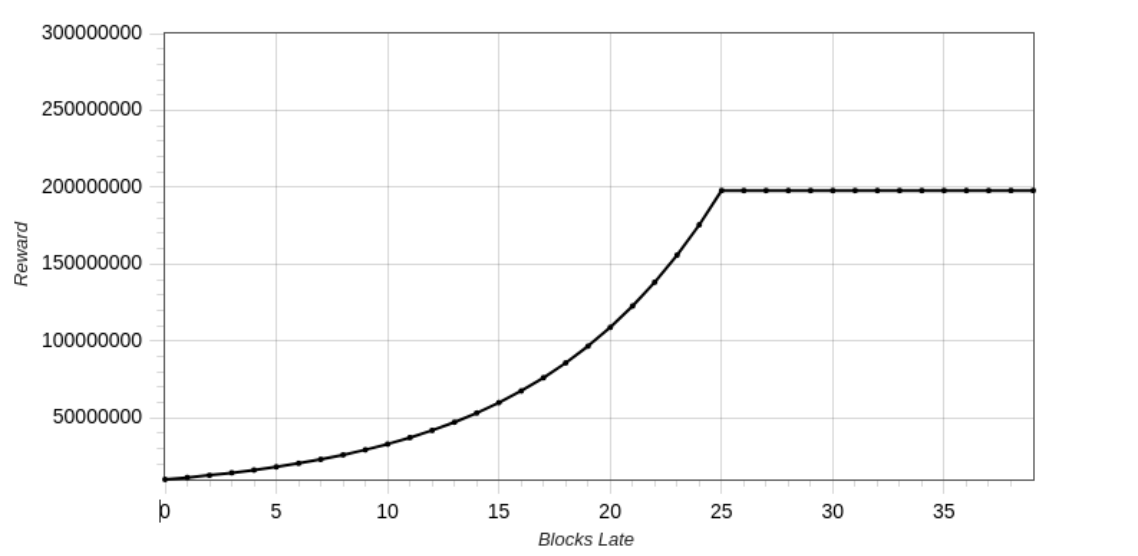
```
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 10000000 when blocksLate: 0
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 11268249 when blocksLate: 1
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 12697344 when blocksLate: 2
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 14307685 when blocksLate: 3
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 16122252 when blocksLate: 4
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 18166924 when blocksLate: 5
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 20470979 when blocksLate: 6
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 23067178 when blocksLate: 7
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 25992581 when blocksLate: 8
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 29289213 when blocksLate: 9
```

MANUAL TESTING

```
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 33003750 when blocksLate: 10
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 37189548 when blocksLate: 11
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 41906066 when blocksLate: 12
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 47220699 when blocksLate: 13
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 53209258 when blocksLate: 14
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 59957117 when blocksLate: 15
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 67560429 when blocksLate: 16
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 76130197 when blocksLate: 17
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 85784950 when blocksLate: 18
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 96663815 when blocksLate: 19
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 108921815 when blocksLate: 20
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 122733509 when blocksLate: 21
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 138304230 when blocksLate: 22
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 155843255 when blocksLate: 23
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 175605779 when blocksLate: 24
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 197882195 when blocksLate: 25
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 197882195 when blocksLate: 26
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 197882195 when blocksLate: 27
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 197882195 when blocksLate: 28
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 197882195 when blocksLate: 29
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 197882195 when blocksLate: 30
Calling -> contract_SeasonFacet.sunrise({'from': user1})
Reward is: 197882195 when blocksLate: 31
```

MANUAL TESTING

```
65 Calling -> contract_SeasonFacet.sunrise({'from': user1})
66 Reward is: 197882195 when blocksLate: 32
67 Calling -> contract_SeasonFacet.sunrise({'from': user1})
68 Reward is: 197882195 when blocksLate: 33
69 Calling -> contract_SeasonFacet.sunrise({'from': user1})
70 Reward is: 197882195 when blocksLate: 34
71 Calling -> contract_SeasonFacet.sunrise({'from': user1})
72 Reward is: 197882195 when blocksLate: 35
73 Calling -> contract_SeasonFacet.sunrise({'from': user1})
74 Reward is: 197882195 when blocksLate: 36
75 Calling -> contract_SeasonFacet.sunrise({'from': user1})
76 Reward is: 197882195 when blocksLate: 37
77 Calling -> contract_SeasonFacet.sunrise({'from': user1})
78 Reward is: 197882195 when blocksLate: 38
79 Calling -> contract_SeasonFacet.sunrise({'from': user1})
80 Reward is: 197882195 when blocksLate: 39
```



THANK YOU FOR CHOOSING

// HALBORN