HALBORN

Beanstalk - Pipeline

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **October 24th, 2022 - November 11th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	26/10/2022	Luis Buendia
0.2	Document Additions	06/11/2022	Gokberk Gulgun
0.3	Document Additions	10/11/2022	Luis Buendia
0.4	Draft Review	11/11/2022	Gabi Urrutia
1.0	Remediation Plan	11/13/2022	Luis Buendia
1.1	Remediation Plan Review	11/15/2022	Gabi Urrutia
1.2	New Scope Update	11/24/2022	Luis Buendia
1.3	New Scope Update Review	11/24/2022	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
Gokberk Gulgun	Halborn	Gokberk.Gulgun@halborn.com
Luis Buendia	Halborn	Luis.Buendia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Beanstalk engaged Halborn to conduct a security audit on their smart contracts beginning on October 24th, 2022 and ending on November 11th, 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 accepted by 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)
- Dynamic Analysis and test coverage. (Foundry)

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.

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

1.4 SCOPE

IN-SCOPE :

PULL REQUEST
01f61ad7e4954d01bc718a9f82c8a833c63ee3e9

The next commit has been added to the audit to reflect the final version that will be deployed.
e5da2c5304a6445242f733ad5bc9a56f9b0396369

ADDITIONAL COMMIT IDS :

bf81f2bd1c4d0f27b6ae1c0c12732dd6c79ded6c

256d83162687eed4d589bbf24e0a61a590c11326

e193bdf747e804c13280453f3dbb52ebc797091b

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	2	6

LIKELIHOOD

IMPACT

(HAL-02)		(HAL-01)		
	(HAL-03)			
(HAL-04) (HAL-05) (HAL-06) (HAL-07) (HAL-08) (HAL-09)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - IMPROPER MEMORY ACCESS	Medium	RISK ACCEPTED
HAL02 - IMPLEMENTATION IS NOT RESISTANT TO SELECTOR COLLUSION	Low	RISK ACCEPTED
HAL03 - pasteAdvancedBytes REVERTS IF NO DATA IS RETURNED	Low	RISK ACCEPTED
HAL04 - OUTDATED SOLIDITY VERSION	Informational	ACKNOWLEDGED
HAL05 - REDUNDANT PAYABLE DEFINITON	Informational	ACKNOWLEDGED
HAL06 - USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS	Informational	ACKNOWLEDGED
HAL07 - MISSING/INCOMPLETE NATSPEC COMMENTS	Informational	ACKNOWLEDGED
HAL08 - REDUNDANT SENDER PARAMETER	Informational	ACKNOWLEDGED
HAL09 - OPTIMIZE UNSIGNED INTEGER COMPARISON	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) IMPROPER MEMORY ACCESS - MEDIUM

Description:

The getEthValue function of the Pipeline.sol contract tries to get the last 32 bytes of the received byte stream. However, if this value is not set, the function will access the next 32 bytes of memory and take the value as if it were the ether that the user wanted to use in this call.

Code Location:

protocol/contracts/pipeline/Pipeline.sol

Listing 1: (Line 111)

```
109 function getEthValue(bytes calldata advancedData) private pure
 ↳ returns (uint256 value) {
110     if (advancedData[1] == 0x00) return 0;
111     assembly { value := calldataload(sub(add(advancedData.offset,
 ↳ advancedData.length), 32))}
112 }
```

Risk Level:

Likelihood - 3
Impact - 3

Proof Of Concept:

The following code snippet produces the behavior described in the source code.

Listing 2

```
1 function testMemoryOutOfBound() public {
2     AdvancedPipe[] memory p = new AdvancedPipe[](1);
3     p[0].advancedData = new bytes(32);
4     p[0].advancedData[1] = 0x01;
5     p[0].advancedData[0] = 0x00;
6     p[0].target = address(mock);
7     bytes[] memory r = pipeline.advancedPipe(p);
8 }
```

As specified in the screenshot below, the transaction reverts with an OutOfFund error when trying to send the ether specified on the value parameter.

```
PS [redacted]beanstalk\Beanstalk\protocol> forge test --match-test testMemoryOutOfBound -vvv --silent

Running 1 test for test/Pipeline.t.sol:PiepelineTest
[FAIL. Reason: EvmError: Revert] testMemoryOutOfBound() (gas: 25483)
Logs:
  Offset:  260
  Len:  32
  Load:  260
  176684706477838432958329750074291851582748389687561895812160620129261977 6

Traces:
  [25483] PiepelineTest::testMemoryOutOfBound()
    ├─ [16477] Pipeline::advancedPipe([(0x185a4dc360ce69bdccee33b3784b0282f7961aea, 0x, 0x0001000000000000000000000000000000000000000000000000000000000000
    │   ├─ [0] console::9710a9d0(0000000000000000000000000000000000000000000000000000000000000040000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000084f66667365743a2000000000000000000000000000000000000000000000000000000000) [staticcall]
    │   │   └─ ← ()
    │   ├─ [0] console::9710a9d0(0000000000000000000000000000000000000000000000000000000000000040000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000054c656e3a2000000000000000000000000000000000000000000000000000000000) [staticcall]
    │   │   └─ ← ()
    │   ├─ [0] console::9710a9d0(0000000000000000000000000000000000000000000000000000000000000040000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000064c6f61643a2000000000000000000000000000000000000000000000000000000000) [staticcall]
    │   │   └─ ← ()
    │   ├─ [0] console::f5b1bba9(0001000000000000000000000000000000000000000000000000000000000000000000) [staticcall]
    │   │   └─ ← ()
    │   ├─ [0] Mock::fallback{value: 176684706477838432958329750074291851582748389687561895812160620129261977 6}()
    │   │   └─ ← "EvmError: OutOfFund"
    │   └─ ← "EvmError: Revert"
    └─ ← "EvmError: Revert"

Test result: FAILED. 0 passed; 1 failed; finished in 3.35ms

Failing tests:
Encountered 1 failing test in test/Pipeline.t.sol:PiepelineTest
[FAIL. Reason: EvmError: Revert] testMemoryOutOfBound() (gas: 25483)

Encountered a total of 1 failing tests, 0 tests succeeded
```

Recommendation:

Control the length and offset value to avoid loading memory values out of the bytes received as input parameter. When using assembly, it is important to control memory accesses, as unexpected behaviors with serious security risks are possible.

Remediation Plan:

RISK ACCEPTED: The Beanstalk team accepted the risk of this finding. They stated that:

- It is caused by an improper off-chain encoding.
- It results in a potential lose of funds only for the user.

3.2 (HAL-02) IMPLEMENTATION IS NOT RESISTANT TO SELECTOR COLLUSION - LOW

Description:

During the code review, It has been noticed that there is no white-listing on the selectors. In the solidity, Multiple functions can have the same signature. For example, these two functions have the same signature:

- gasprice_bit_ether(int128)
- transferFrom(address,address,uint256)

Code Location:

protocol/contracts/pipeline/Pipeline.sol

Listing 3

```
    function _farm(bytes calldata data) private returns (bytes
↳ memory result) {
        bytes4 selector; bool success;
        assembly { selector := calldataload(data.offset) }
        address facet = LibFunction.facetForSelector(selector);
        (success, result) = facet.delegatecall(data);
        LibFunction.checkReturn(success, result);
    }

    // delegatecall a Beanstalk function using memory data
    function _farmMem(bytes memory data) private returns (bytes
↳ memory result) {
        bytes4 selector; bool success;
        assembly { selector := mload(add(data, 32)) }
        address facet = LibFunction.facetForSelector(selector);
        (success, result) = facet.delegatecall(data);
        LibFunction.checkReturn(success, result);
    }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Consider adding whitelist in function selectors.

Remediation Plan:

RISK ACCEPTED: The Beanstalk team accepted the risk of this finding. They stated that:

- It is caused by an improper off-chain encoding
- It results in a potential lose of funds only for the user.

3.3 (HAL-03) pasteAdvancedBytes REVERTS IF NO DATA IS RETURNED - LOW

Description:

The pasteAdvancedBytes function in the LibFunction.sol contract attempts to copy the returnData to the pastedData variable. However, if returnData does not match with callData, the functions revert without handling the error or providing a proper message.

Code Location:

protocol/contracts/libraries/LibFunction.sol

Listing 4: (Line 113)

```
107     function pasteAdvancedBytes(
108         bytes memory callData,
109         bytes[] memory returnData,
110         bytes32 copyParams
111     ) internal view returns (bytes memory pastedData) {
112         // Shift `copyParams` right 22 bytes to insolated
 ↳ reduceDataIndex
113         bytes memory copyData = returnData[uint256((copyParams <<
 ↳ 16) >> 176)];
114         pastedData = paste32Bytes(
115             copyData,
116             callData,
117             uint256((copyParams << 96) >> 176), // Isolate
 ↳ copyIndex
118             uint256((copyParams << 176) >> 176) // Isolate
 ↳ pasteIndex
119         );
120     }
```

Risk Level:

Likelihood - 2
Impact - 2

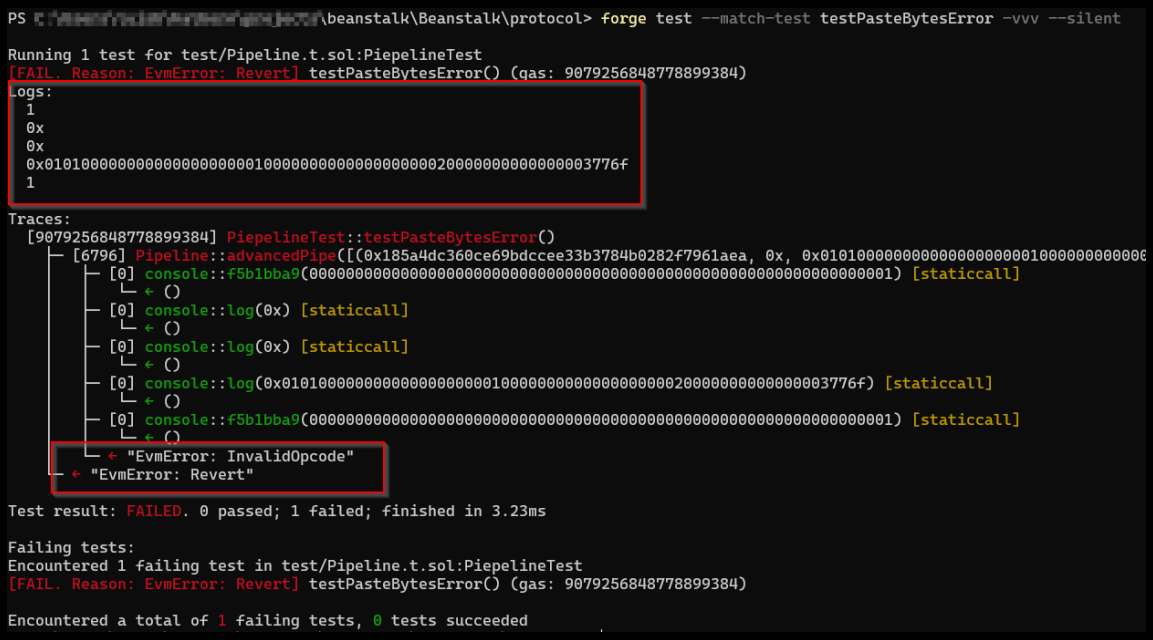
Proof Of Concept:

The following code snippet produces the behavior described in the source code.

Listing 5

```
function testPasteBytesError() public {
    bytes2 prebytes = 0x0101;
    uint80 a = 1;
    uint80 b = 2;
    uint64 c = 3;
    string memory d = 'wololo';
    bytes4 e = 0x0a0b0c0d;
    bool f = false;
    bytes memory data = abi.encodePacked(prebytes,a,b,c,d,e,f);
    AdvancedPipe[] memory p = new AdvancedPipe[](1);
    p[0].advancedData = data;
    p[0].target = address(mock);
    bytes[] memory r = pipeline.advancedPipe(p);
}
```

For clarity, some logs have been added to the console with the different parameter values when entering the pastedData function. The following screenshot shows the tracking when reverting, as well as the values.



The modified source code that matches the screenshot above is below.

Listing 6

```
function pasteAdvancedBytes(
    bytes memory callData,
    bytes[] memory returnData,
    bytes32 copyParams
) internal view returns (bytes memory pastedData) {
    // Shift `copyParams` right 22 bytes to insolated
↳ reduceDataIndex
    console.log(returnData.length);
    console.logBytes(returnData[0]);
    console.logBytes(callData);
    console.logBytes32(copyParams);
    console.log(uint256((copyParams << 16) >> 176));
    bytes memory copyData = returnData[uint256((copyParams << 16)
↳ >> 176)];
    pastedData = paste32Bytes(
        copyData,
        callData,
        uint256((copyParams << 96) >> 176), // Isolate copyIndex
        uint256((copyParams << 176) >> 176) // Isolate pasteIndex
    );
}
```

Recommendation:

Validate input parameters to avoid reverting under unexpected conditions. For this case, if the index to be accessed is greater than or equal to the length of the byte array, revert instead of trying to access it. Or handle the error to continue execution.

Remediation Plan:

RISK ACCEPTED: The Beanstalk team accepted the risk of this finding. They stated that:

- It is caused by an improper off-chain encoding
- It results in a potential lose of funds only for the user.

3.4 (HAL-04) OUTDATED SOLIDITY VERSION - INFORMATIONAL

Description:

The current version of Solidity is outdated. The system uses version 0.7.6. This version is known to be risky, as it still allows overflow issues when performing basic arithmetic operations. But the feature that is more interesting is the new Memory Safe for inline assembly.

This feature was introduced in Solidity version 0.8.16. This helps to respect Solidity's memory model while using inline assembly Yul code.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Update to a robust Solidity version that offers more security features and can help protect and optimize the code. Reference, solidity documentation.

Remediation Plan:

ACKNOWLEDGED: The Beanstalk team acknowledged the finding.

3.5 (HAL-05) REDUNDANT PAYABLE DEFINITION - INFORMATIONAL

Description:

In the Pipeline contract, the **multiPipe** function makes a list of calls to external functions without sending ether. The function is marked as a payable. Adding payable actually decreases the compiled bytecode size of your functions. Because without payable, the compiler needs to verify that the **msg.value** of the transaction is equal to 0 and will add the following opcode snippet to your bytecode.

Code Location:

protocol/contracts/pipeline/Pipeline.sol

Listing 7

```
1     function multiPipe(Pipe[] calldata pipes)
2         external
3         payable
4         override
5         returns (bytes[] memory results)
6     {
7         results = new bytes[](pipes.length);
8         for (uint256 i = 0; i < pipes.length; i++) {
9             results[i] = _pipe(pipes[i].target, pipes[i].data, 0);
10        }
11    }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Remove payable from the function.

Remediation Plan:

ACKNOWLEDGED: The Beanstalk team acknowledged the finding.

3.6 (HAL-06) USE CUSTOM ERRORS INSTEAD OF REVERT STRINGS TO SAVE GAS - INFORMATIONAL

Description:

Custom errors are available from Solidity version 0.8.4. Custom errors save ~50 gas each time they are hit by avoiding having to allocate and store the revert string. Not defining strings also saves deployment gas.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider replacing all revert strings with custom errors.

Remediation Plan:

ACKNOWLEDGED: The Beanstalk team acknowledged the finding.

3.7 (HAL-07) MISSING/INCOMPLETE NATSPEC COMMENTS - INFORMATIONAL

Description:

Functions are missing @param for some of their parameters. Since **NatSpec** is an important part of the code documentation, this affects the understandability, auditability, and usability of the code.

Code Location:

Contracts

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding full **NatSpec** comments so that all functions have full code documentation for future use.

Remediation Plan:

ACKNOWLEDGED: The Beanstalk team acknowledged the finding.

3.8 (HAL-08) REDUNDANT SENDER PARAMETER - INFORMATIONAL

Description:

The `sender == msg.sender` check is redundant. It is not used in the `transferDeposit` function.

Code Location:

protocol/contracts/pipeline/Pipeline.sol

Listing 8

```
1     function transferDeposit(
2         address sender,
3         address recipient,
4         address token,
5         uint32 season,
6         uint256 amount
7     ) external payable returns (uint256 bdv) {
8         require(sender == msg.sender, "invalid sender");
9         bdv = beanstalk.transferDeposit(msg.sender, recipient,
↳ token, season, amount);
10    }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider removing the redundant variable.

Remediation Plan:

ACKNOWLEDGED: The Beanstalk team acknowledged the finding.

3.9 (HAL-09) OPTIMIZE UNSIGNED INTEGER COMPARISON - INFORMATIONAL

Description:

The check != 0 costs less gas compared to > 0 for unsigned integers in require statements with the optimizer enabled. While it may seem like > 0 is cheaper than !=0, this is only true without the optimizer enabled and outside a require statement. If the optimizer is enabled on 10k, and it is on a require statement, it would be more gas efficient.

Code Location:

Listing 9

```
1     modifier withEth() {
2         if (msg.value > 0) s.isFarm = 2;
3         _;
4         if (msg.value > 0) {
5             s.isFarm = 1;
6             LibEth.refundEth();
7         }
8     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Change > 0 compared to != 0.

Remediation Plan:

ACKNOWLEDGED: The Beanstalk team acknowledged the finding.

THANK YOU FOR CHOOSING

// HALBORN