HALBORN

Staderlabs - BnbX

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **June 29th, 2022 - July 5th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 3
CONTACTS 3
1 EXECUTIVE OVERVIEW 4
1.1 INTRODUCTION 5
1.2 AUDIT SUMMARY 5
1.3 TEST APPROACH & METHODOLOGY 5
RISK METHODOLOGY 6
1.4 SCOPE 8
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 9
3 FINDINGS & TECH DETAILS 10
3.1 (HAL-01) INTEGER UNDERFLOW - HIGH 12
Description 12
Code Location 12
Risk Level 14
Recommendation 14
Remediation Plan 14
3.2 (HAL-02) CALLING increaseTotalRedelegated BEFORE USER DEPOSITS MAY CAUSE USER GETTING 0 BNBX - MEDIUM 15
Description 15
Scenario 15
Code Location 15
Risk Level 16
Recommendation 16
Remediation Plan 16
3.3 (HAL-03) CONTRACTS ARE NOT USING disableInitializers FUNCTION - LOW 17

Description 17
Risk Level 17
Recommendation 17
Remediation Plan 17
3.4 (HAL-04) MISSING ADDRESS CHECK - INFORMATIONAL 18
Description 18
Code Location 18
Risk Level 18
Recommendation 18
Remediation Plan 18
4 AUTOMATED TESTING 18
4.1 STATIC ANALYSIS REPORT 20
Description 20
Slither results 20

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	07/05/2022	Pawel Bartunek
0.2	Draft Review	07/05/2022	Gabi Urrutia
1.0	Remediation Plan	07/12/2022	Pawel Bartunek
1.1	Remediation Plan Review	07/13/2022	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
Pawel Bartunek	Halborn	Pawel.Bartunek@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Staderlabs engaged Halborn to conduct a security audit on their smart contracts beginning on June 29th, 2022 and ending on July 5th, 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 one week 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 addressed by the Staderlabs team. The main ones are the following:

- Ensure that increasing staked amount via `increaseTotalRedelegated` does not cause arithmetic issues in other functions.
- Verify BNB/BNBX conversion logic, to ensure users are getting a share after deposit.

1.3 TEST APPROACH & METHODOLOGY

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

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

- Research into architecture and purpose
- Smart contract manual code review and walk-through
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Local deployment (Brownie, Hardhat, 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.

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

1.4 SCOPE

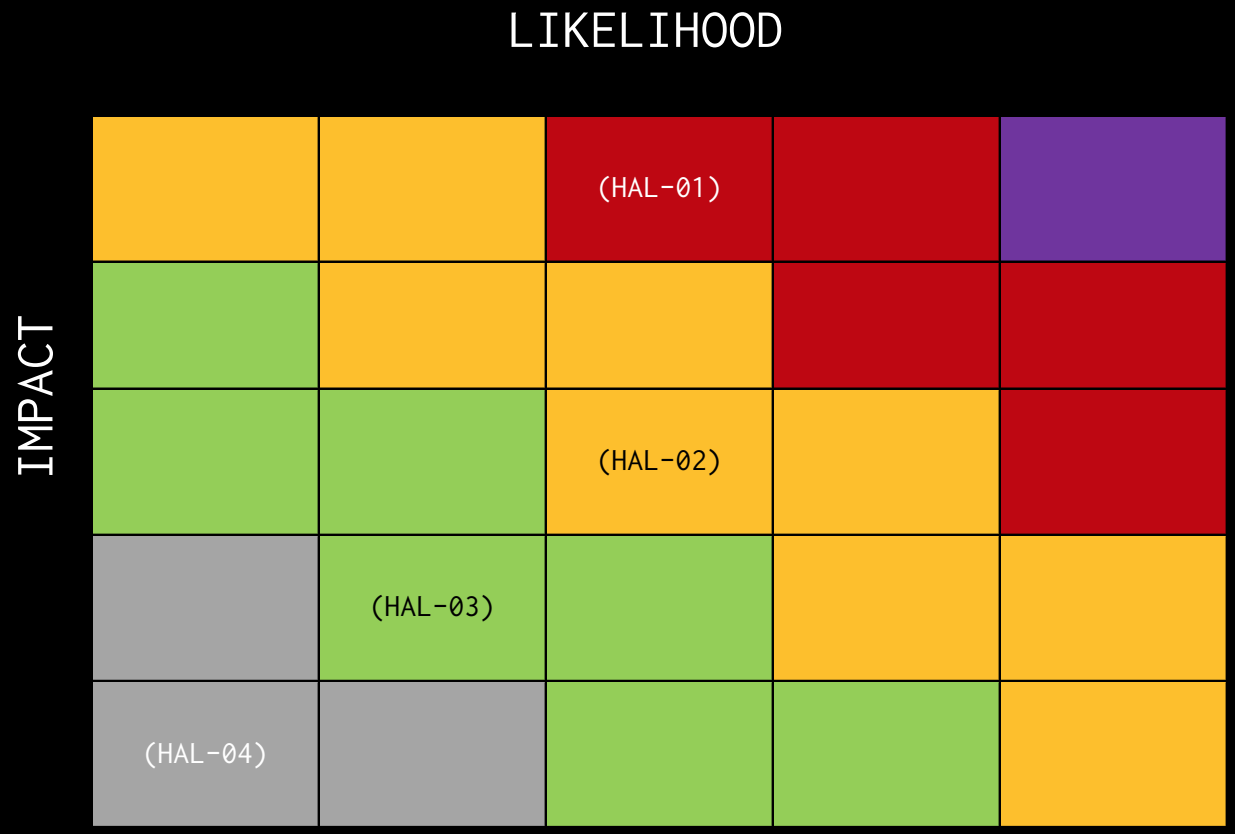
The security assessment was scoped to the following smart contracts:

- BnbX.sol
- StakeManager.sol
- IBnbX.sol
- IStakeManager.sol

Commit ID: 2ddf3e2c30321587742630de90a1414434ff256f
Remediation commit ID d56ab580231c56531edbb780387e1c711236c85d

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	1	1	1



EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL01 - INTEGER UNDERFLOW	High	SOLVED - 07/12/2022
HAL02 - CALLING increaseTotalRedelegated BEFORE USER DEPOSITS MAY CAUSE USER GETTING 0 BNBX	Medium	SOLVED - 07/12/2022
HAL03 - CONTRACTS ARE NOT USING disableInitializers FUNCTION	Low	SOLVED - 07/12/2022
HAL04 - MISSING ADDRESS CHECK	Informational	SOLVED - 07/12/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) INTEGER UNDERFLOW - HIGH

Description:

Calling the increaseTotalRedelegated function is causing an integer underflow in the startUndelegation function.

If the totalRedelegated amount is increased by calling increaseTotalRedelegated, the startUndelegation transaction will revert with: Arithmetic operation underflowed or overflowed outside an unchecked block error (in Solidity > 0.8).
Because the undelegation process will fail, the user will not be able to withdraw the deposited BNB.

Code Location:

The startUndelegation function performs a subtraction:

Listing 1: StakeManager.sol (Line 282)

```
265 function startUndelegation()
266     external
267     override
268     whenNotPaused
269     onlyRole(BOT)
270     returns (uint256 _uuid, uint256 _amount)
271 {
272     require(totalBnbToWithdraw > 0, "No Request to withdraw");
273
274     _uuid = undelegateUUID++;
275     _amount = totalBnbToWithdraw;
276     uuidToBotUndelegateRequestMap[_uuid] = BotUndelegateRequest(
277         block.timestamp,
278         0,
279         _amount
280     );
281
282     totalDeposited -= _amount;
```

```
283     uint256 bnbXToBurn = totalBnbXToBurn; // To avoid Reentrancy
 ↳ attack
284     totalBnbXToBurn = 0;
285     totalBnbToWithdraw = 0;
286
287     IBnbX(bnbX).burn(address(this), bnbXToBurn);
288 }
```

totalDeposited is an amount of BNB deposited and _amount is the value of totalBnbToWithdraw that is calculated in the requestWithdraw function:

Listing 2: StakeManager.sol (Lines 193,206)

```
191 function requestWithdraw(uint256 _amount) external override
 ↳ whenNotPaused {
192     require(_amount > 0, "Invalid Amount");
193     uint256 amountInBnb = convertBnbXToBnb(_amount);
194
195     IERC20Upgradeable(bnbX).safeTransferFrom(
196         msg.sender,
197         address(this),
198         _amount
199     );
200     uint256 totalStakedBnb = getTotalStakedBnb();
201     require(
202         amountInBnb <= (totalStakedBnb - totalBnbToWithdraw),
203         "Not enough BNB to withdraw"
204     );
205
206     totalBnbToWithdraw += amountInBnb;
207     totalBnbXToBurn += _amount;
208     userWithdrawalRequests[msg.sender].push(
209         WithdrawalRequest(undelegateUUID, amountInBnb, block.
 ↳ timestamp)
210     );
211
212     emit RequestWithdraw(msg.sender, _amount, amountInBnb);
213 }
```

The value taken from convertBnbXToBnb is added to totalBnbToWithdraw. The convertBnbXToBnb function calculates its value based on the output

of getTotalPooledBnb:

Listing 3: StakeManager.sol (Line 455)

```
455 uint256 totalPooledBnb = getTotalPooledBnb();
```

Which is using totalRedelegated set by the increaseTotalRedelegated function:

Listing 4: StakeManager.sol (Line 348)

```
347 function getTotalPooledBnb() public view override returns (uint256
 ↳ ) {
348     return (totalDeposited + totalRedelegated);
349 }
```

Risk Level:

Likelihood - 3
Impact - 5

Recommendation:

Make sure that calculations performed by other functions are not affected by increasing the delegated/staked amount with increaseTotalRedelegated.

Remediation Plan:

SOLVED: The addRestakingRewards function (previously called increaseTotalRedelegated), now contains a check: the amount delegated must be greater than 0 to increase. Also, the startUndelegation function recalculates the BnbX/BNB ratio, instead of using a previously calculated one.

3.2 (HAL-02) CALLING increaseTotalRedelegated BEFORE USER DEPOSITS MAY CAUSE USER GETTING 0 BNBX - MEDIUM

Description:

If the increaseTotalRedelegated function is called when totalDeposited is 0 and the increased amount is greater than the amount of assets deposited by the first user, the depositor will get 0 BnbX tokens.

Scenario:

- Deposited amount is 0 (none deposited yet)
- increaseTotalRedelegated function is called with 10 BNB as parameter
- User deposits 1 BNB

Result:
The user gets 0 BnbX in return for a deposit of 1 BNB.

Code Location:

Listing 5: StakeManager.sol (Line 435)

```
426 function convertBnbToBnbX(uint256 _amount)
427     public
428     view
429     override
430     returns (uint256)
431 {
432     uint256 totalShares = IBnbX(bnbX).totalSupply();
433     totalShares = totalShares == 0 ? 1 : totalShares;
434
435     uint256 totalPooledBnb = getTotalPooledBnb();
436     totalPooledBnb = totalPooledBnb == 0 ? 1 : totalPooledBnb;
437
```

```
438     uint256 amountInBnbX = (_amount * totalShares) /
 ↳ totalPooledBnb;
439
440     return amountInBnbX;
441 }
```

`getTotalPooledBnb` calculation:

Listing 6: StakeManager.sol (Line 348)

```
347 function getTotalPooledBnb() public view override returns (uint256
 ↳ ) {
348     return (totalDeposited + totalRedelegated);
349 }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Consider changing the conversion calculation logic so that increasing the amount delegated/staked has less impact on first deposits.

Remediation Plan:

SOLVED: The `addRestakingRewards` (previously called as `increaseTotalRedelegated`) function, now contains a check: the amount delegated must be greater than `0` to increase. This solves the problem of first depositor getting `0` BnbX tokens: Reference

3.3 (HAL-03) CONTRACTS ARE NOT USING disableInitializers FUNCTION - LOW

Description:

The StakeManager and BnbX contracts use Open Zeppelin Initializable module. According to the Open Zeppelin guidelines the _disableInitializers function call should be added to the constructor to lock contracts after deployment.

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider calling the _disableInitializers function in the contract's constructor:

Listing 7

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

Remediation Plan:

SOLVED: The constructor with call to _disableInitializers() was added to StakeManager and BnbX contracts.

3.4 (HAL-04) MISSING ADDRESS CHECK - INFORMATIONAL

Description:

The lack of zero address validation has been found in many instances when assigning user-supplied address values to state variables directly.

Code Location:

BnbX.sol, #37-49
The setStakeManager function allows you to set a stakeManager address to 0x0.

StakeManager.sol, #70-73
The StakeManager's contract initialization function does not check that the passed addresses are non-0.

StakeManager.sol, #326-338
The setBotAddress function allows a bot's address to be set to 0x0.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

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

Remediation Plan:

SOLVED: Added zero address checks in commit 4e04e46729153b6cb50d2ce4da2f807611fcc4d8

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

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

Slither results:

StakeManager.sol

Listing 8

```
StakeManager.startDelegation() (contracts/StakeManager.sol
↳ #103-142) ignores return value by ITokenHub(tokenHub).transferOut{
↳ value: (amount + relayFeeReceived)}(address(0),bcDepositWallet,
↳ amount,expireTime) (contracts/StakeManager.sol#131-136)
Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#unused-return

StakeManager.initialize(address,address,address,address,address).
↳ _bnbX (contracts/StakeManager.sol#58) lacks a zero-check on :
        - bnbX = _bnbX (contracts/StakeManager.sol#70)
StakeManager.initialize(address,address,address,address,address).
↳ _tokenHub (contracts/StakeManager.sol#60) lacks a zero-check on :
        - tokenHub = _tokenHub (contracts/StakeManager.sol#71)
StakeManager.initialize(address,address,address,address,address).
↳ _bcDepositWallet (contracts/StakeManager.sol#61) lacks a zero-
↳ check on :
        - bcDepositWallet = _bcDepositWallet (contracts/
↳ StakeManager.sol#72)
StakeManager.initialize(address,address,address,address,address).
↳ _bot (contracts/StakeManager.sol#62) lacks a zero-check on :
        - bot = _bot (contracts/StakeManager.sol#73)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in StakeManager.requestWithdraw(uint256) (contracts/StakeManager.sol#191-213):
    External calls:
    - IERC20Upgradeable(bnbX).safeTransferFrom(msg.sender,address(this),_amount) (contracts/StakeManager.sol#195-199)
    State variables written after the call(s):
    - totalBnbToWithdraw += amountInBnb (contracts/StakeManager.sol#206)
    - totalBnbXToBurn += _amount (contracts/StakeManager.sol#207)
    - userWithdrawalRequests[msg.sender].push(WithdrawalRequest(undelegateUUID,amountInBnb,block.timestamp)) (contracts/StakeManager.sol#208-210)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in StakeManager.claimWithdraw(uint256) (contracts/StakeManager.sol#215-235):
    External calls:
    - AddressUpgradeable.sendValue(address(user),amount) (contracts/StakeManager.sol#232)
    Event emitted after the call(s):
    - ClaimWithdrawal(user,_idx,amount) (contracts/StakeManager.sol#234)
Reentrancy in StakeManager.requestWithdraw(uint256) (contracts/StakeManager.sol#191-213):
    External calls:
    - IERC20Upgradeable(bnbX).safeTransferFrom(msg.sender,address(this),_amount) (contracts/StakeManager.sol#195-199)
    Event emitted after the call(s):
    - RequestWithdraw(msg.sender,_amount,amountInBnb) (contracts/StakeManager.sol#212)
Reentrancy in StakeManager.startDelegation() (contracts/StakeManager.sol#103-142):
    External calls:
    - ITokenHub(tokenHub).transferOut{value: (amount + relayFeeReceived)}(address(0),bcDepositWallet,amount,expireTime) (contracts/StakeManager.sol#131-136)
    Event emitted after the call(s):
    - TransferOut(amount) (contracts/StakeManager.sol#138)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

```
39
40 StakeManager.completeDelegation(uint256) (contracts/StakeManager.
↳ sol#149-166) uses timestamp for comparisons
41     Dangerous comparisons:
42     - require(bool,string)((uuidToBotDelegateRequestMap[_uuid].
↳ amount > 0) && (uuidToBotDelegateRequestMap[_uuid].endTime == 0),
↳ Invalid UUID) (contracts/StakeManager.sol#155-159)
43 StakeManager.claimWithdraw(uint256) (contracts/StakeManager.sol
↳ #215-235) uses timestamp for comparisons
44     Dangerous comparisons:
45     - require(bool,string)(uuidToBotUndelegateRequestMap[uuid].
↳ endTime != 0,Not able to claim yet) (contracts/StakeManager.sol
↳ #225-228)
46 StakeManager.isClaimable(address,uint256) (contracts/StakeManager.
↳ sol#243-257) uses timestamp for comparisons
47     Dangerous comparisons:
48     - _isClaimable = (uuidToBotUndelegateRequestMap[uuid].endTime
↳ != 0) (contracts/StakeManager.sol#256)
49 StakeManager.completeUndelegation(uint256) (contracts/StakeManager
↳ .sol#297-318) uses timestamp for comparisons
50     Dangerous comparisons:
51     - require(bool,string)((uuidToBotUndelegateRequestMap[_uuid].
↳ amount > 0) && (uuidToBotUndelegateRequestMap[_uuid].endTime == 0)
↳ ,Invalid UUID) (contracts/StakeManager.sol#304-308)
52     - require(bool,string)(amount == uuidToBotUndelegateRequestMap
↳ [_uuid].amount,Incorrect Amount of Fund) (contracts/StakeManager.
↳ sol#311-314)
53 Reference: https://github.com/crytic/slither/wiki/Detector-
↳ Documentation#block-timestamp
54 contracts/StakeManager.sol analyzed (15 contracts with 57
↳ detectors), 13 result(s) found
```

BnbX.sol:

Listing 9

```
1 contracts/BnbX.sol analyzed (13 contracts with 57 detectors), 0
↳ result(s) found
```

As a result of the tests carried out with the Slither tool, some results were obtained and reviewed by Halborn. Based on the results reviewed, some vulnerabilities were determined to be false positives.

THANK YOU FOR CHOOSING

// HALBORN