// HALBORN

BSCEX LAUNCHPOOLx

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **March 14, 2021**

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 6
RISK METHODOLOGY 6
1.4 SCOPE 8
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 9
3 FINDINGS & TECH DETAILS 10
3.1 (HAL-01) FLOATING PRAGMA - LOW 12
Description 12
Code Location 12
Risk Level 12
Recommendations 12
3.2 (HAL-02) PRAGMA VERSION DEPRECATED - LOW 13
Description 13
Code Location 13
Risk Level 13
Recommendations 13
3.3 (HAL-03) FOR LOOP OVER DYNAMIC ARRAY - LOW 14
Description 14
Code Location 14
Risk Level 15

Recommendations 15

3.4 (HAL-04) IGNORE RETURN VALUES - LOW 15

Description 15

Code Location 15

Risk Level 16

Recommendation 16

3.5 (HAL-05) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL 16

Description 16

Code Location 17

Risk Level 18

Recommendations 18

3.6 STATIC ANALYSIS REPORT 18

Description 18

Results 19

3.7 AUTOMATED SECURITY SCAN 22

Description 22

Results 22

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	03/17/2021	Nishit Majithia
0.2	Document Edits	03/17/2021	Gabi Urrutia
1.0	Final Version	03/18/2021	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Nishit Majithia	Halborn	nishit.majithia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

BSCEX is a decentralized non-custodial cryptocurrency exchange that aims to facilitate the services that exchange-centered ecosystems provide. This project's primary goal is to bring Binance's off-chain services on-chain to the Binance Smart Chain (BSC). LaunchpoolX is the on-chain version of Binance exchange's Launchpool.

The security assessment was scoped to the smart contract BSCXNTS.sol. An audit of the security risk and implications regarding the changes introduced by the development team at BSCEX prior to its production release shortly following the assessments deadline.

Though the outcome of this security audit is satisfactory; due to time and resource constraints, only testing and verification of essential properties were performed to achieve objectives and deliverable set in the scope. It is important to remark the use of the best practices for secure smart contract development.

1.2 AUDIT SUMMARY

The team at Halborn was provided a week timeframe for the engagement and assigned one full time security engineers to audit the security of the smart contract. The security engineer is blockchain and smart contract security subject matter experts, with experience in advanced penetration testing, smart contract hacking, and have a deep knowledge in multiple blockchain protocols.

The purpose of this audit to achieve the following:
- Ensure that smart contract functions are intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified 5 security risks, and recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated

for expected logic and state.

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 the smart contract audit.
While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts 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 read 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.
- Scanning of solidity files for vulnerabilities, security hotspots, or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Truffle, Ganache)
- Smart Contract Fuzzing and dynamic state exploitation (Echidna) Symbolic Execution / EVM bytecode security assessment (limited time)

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. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics

that was 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:
The security assessment was scoped to the smart contract:
- BSCXNTS.sol
Specific commit of contract: commit 1c4a8df4ba1c0051c0a705dda006844756afe6ea

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

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	4	1

LIKELIHOOD

IMPACT

(HAL-01) (HAL-02)	(HAL-03)			
		(HAL-04)		
(HAL-05)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
FLOATING PRAGMA	Low	-
PRAGMA VERSION DEPRECATED	Low	-
FOR LOOP OVER DYNAMIC ARRAY	Low	-
IGNORE RETURN VALUES	Low	-
POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational	-
STATIC ANALYSIS	-	-
AUTOMATED SECURITY SCAN	-	-

FINDINGS & TECH DETAILS

3.1 (HAL-01) FLOATING PRAGMA - LOW

Description:

All Smart Contracts use the floating pragma ^0.6.0. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an updated compiler version that might introduce bugs or discovered vulnerabilities in the newest versions that affect the contract system negatively.

Code Location:

BSCXNTS.sol: [Line #2]

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.6.0;
3
4  import './interfaces/IERC20.sol';
5  import './libraries/SafeERC20.sol';
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendations:

Consider lock the pragma version known bugs for the compiler version. When possible, do not use floating pragma in the final live deployment. Pragma can also be locked fixing the compiler version in the configuration file when you deploy contracts with truffle or hardhat frameworks.

3.2 (HAL-02) PRAGMA VERSION DEPRECATED - LOW

Description:

The current version in use for the contracts is pragma 0.6.0. While this version is still functional, and most security issues safely implemented by mitigating contracts with other utility contracts such as SafeMath.sol and ReentrancyGuard.sol, the risk to the long-term sustainability and integrity of the solidity code increases.

Code Location:

BSCXNTS.sol: [Line #2]

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.6.0;
3
4  import './interfaces/IERC20.sol';
5  import './libraries/SafeERC20.sol';
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendations:

At the time of this audit, the current version is already at 0.8.2. When possible, use the most updated and tested pragma versions to take advantage of new features that provide checks and accounting, as well as prevent insecure use of code. (0.6.12)

3.3 (HAL-03) FOR LOOP OVER DYNAMIC ARRAY - LOW

Description:

When smart contracts are deployed or functions inside them are called, the execution of these actions always requires a certain amount of gas, based on how much computation is needed to complete them. Programming patterns that are harmless in centralized applications can lead to Denial of Service conditions in smart contracts when the cost of executing a function exceeds the block gas limit. Modifying an array of unknown size, that increases in size over time, can lead to such a Denial of Service condition.

A situation in which the block gas limit can be an issue is in sending funds to an array of addresses. Even without any malicious intent, this can easily go wrong.

Code Location:

BSCXNTS.sol: [Line #252]

```
247         uint256 _startBlock
248     ) public pure returns (uint256) {
249         uint256 result = 0;
250         if (_from < _startBlock) return 0;
251
252         for (uint256 i = 0; i < _halvingAtBlock.length; i++) {
253             uint256 endBlock = _halvingAtBlock[i];
254
255             if (_to <= endBlock) {
256                 uint256 m = _to.sub(_from).mul(_rewardMultiplier[i]);
257                 return result.add(m);
258             }
```

Dynamic array _halvingAtBlock is in control of caller and not bounded by any value. So in this case, dynamic array length can e anything. This array can go up to very large uint256 value, which is very large to exhaust the gas value.

Risk Level:

Likelihood - 2
Impact - 3

Recommendations:

Actions that require looping across the entire data structure should be avoided. If you absolutely must loop over an array of unknown size, then you should plan for it to potentially take multiple blocks, and therefore require multiple transactions. In this case, if you want loop over _halvingAtBlock then the size of _halvingAtBlock should be restricted.

3.4 (HAL-04) IGNORE RETURN VALUES - LOW

Description:

The return value of an external call is not stored in a local or state variable. In contract BSCXNTS.sol, there are few instances where external methods are being called and return value(bool) are being ignored.

Code Location:

BSCXNTS.sol: Line #224, Line #230, Line #233, Line #352, Line #366, Line #381, Line #514

```
220         uint256 forFarmer;
221         (forBurn, forDev, forFarmer) = getPoolReward(_pid);
222
223         if (forBurn > 0) {
224             pool.rewardToken.burn(forBurn);
225         }
226
227         if (forDev > 0) {
228             uint256 lockAmount = forDev.mul(pool.percentLockReward).div(100);
229             if (teamAddresses[_pid] != address(0)) {
230                 pool.rewardToken.transfer(teamAddresses[_pid], forDev.sub(lockAmount));
231                 farmLock(teamAddresses[_pid], lockAmount, _pid);
232             } else {
233                 pool.rewardToken.transfer(devaddr, forDev.sub(lockAmount));
234                 farmLock(devaddr, lockAmount, _pid);
235             }
236         }
237         pool.accRewardPerShare = pool.accRewardPerShare.add(forFarmer.mul(1e12).div(lpSupply));
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Add return value check to avoid unexpected crash of the contract. Return value check will help in handling the exceptions better way.

3.5 (HAL-05) POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In smart contracts Ownable.sol and BSCXNTS.sol, there are many methods like owner(), renounceOwnership(), transferOwnership(), deposit(), totalLockInPool(), setPercentLPLevelRefer(), setAmountLPStakeLevelRefer(), setReferralLPToken(), setStatus(), setTeamAddressPool(), dev(), lastUnlockBlock(), unlock(), set(), add(), withdraw(), getNewRewardPerBlock(), claimReward(), emergencyWithdraw(), lockOf() and totalLock() are marked as public but they are never directly called by another function in the same contract or in any of its descendants,

so better to mark these methods as external. In addition, external functions are cheaper than public functions in gas costs.

Code Location:

Ownable.sol: Line #22, Line #41, Line #50

```
21      */
22      function owner() public view returns (address) {
23          return _owner;
24      }
25
26      /**
27       * @dev Throws if called by any account other than the owner.
28       */
29      modifier onlyOwner() {
30          require(_owner == _msgSender(), "Ownable: caller is not the owner");
31          _;
32      }
33
34      /**
35       * @dev Leaves the contract without owner. It will not be possible to call
36       * `onlyOwner` functions anymore. Can only be called by the current owner.
37       *
38       * NOTE: Renouncing ownership will leave the contract without an owner,
39       * thereby removing any functionality that is only available to the owner.
40       */
41      function renounceOwnership() public virtual onlyOwner {
42          emit OwnershipTransferred(_owner, address(0));
43          _owner = address(0);
44      }
45
46      /**
47       * @dev Transfers ownership of the contract to a new account (`newOwner`).
48       * Can only be called by the current owner.
49       */
50      function transferOwnership(address newOwner) public virtual onlyOwner {
51          require(newOwner != address(0), "Ownable: new owner is the zero address");
52          emit OwnershipTransferred(_owner, newOwner);
```

BSCXNTS.sol: Line #105, Line #156, Line #161, Line #167, Line #173, Line #179, Line #186, Line #307, Line #387, Line #410, Line #429, Line #439, Line #444, Line #455, Line #459, Line #463, Line #467, Line #504

```
155
156     function setStatus(bool _status) public onlyOwner {
157         status = _status;
158         emit Status(msg.sender, status);
159     }
160
161     function setReferralLPToken(IERC20 _referralLPToken) public onlyOwner {
162         referralLPToken = _referralLPToken;
163         emit ReferralLPToken(msg.sender, referralLPToken);
164     }
165
166     // Set team address receive reward
167     function setTeamAddressPool(uint256 _pid, address _teamAddress) public {
168         require(msg.sender == teamAddresses[_pid], "dev: wut?");
169         teamAddresses[_pid] = _teamAddress;
170         emit TeamAddressPool(msg.sender, _pid, teamAddresses[_pid]);
171     }
172
173     function setAmountLPStakeLevelRefer(uint256 _stakeAmountLPLv1, uint256 _stakeAmountLPLv2) public onlyOwner {
174         stakeAmountLPLv1 = _stakeAmountLPLv1;
175         stakeAmountLPLv2 = _stakeAmountLPLv2;
176         emit AmountLPStakeLevelRefer(msg.sender, stakeAmountLPLv1, stakeAmountLPLv2);
177     }
178
179     function setPercentLPLevelRefer(uint256 _percentForReferLv1, uint256 _percentForReferLv2) public onlyOwner {
180         percentForReferLv1 = _percentForReferLv1;
181         percentForReferLv2 = _percentForReferLv2;
182         emit PercentLPLevelRefer(msg.sender, percentForReferLv1, percentForReferLv2);
183     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

If the function is not intended to call internally nor by their descendants as well then it is better to mark all these functions as external instead of public to save some gas.

3.6 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. 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. 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.

Results:

```
INFO:Detectors:
BSCXNTS (contracts/BSCXNTS.sol#9-520) contract sets array length with a user-controlled value:
        - halvingAtBlocks[_pid].push(halvingAtBlock) (contracts/BSCXNTS.sol#149)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#array-length-assignment
```

Issue already mentioned above regarding looping over unbounded data structure.

```
INFO:Detectors:
BSCXNTS.getPoolReward(uint256) (contracts/BSCXNTS.sol#270-290) performs a multiplication on the result of a division:
        -amount = multiplier.mul(pool.rewardPerBlock).mul(pool.allocPoint).div(totalAllocPoints[pool.rewardToken]) (contracts/BSCXNTS.sol#274)
        -forBurn = amount.mul(pool.percentForBurn).div(100) (contracts/BSCXNTS.sol#286)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```

It is false positive since multiplication is being done to percentage value.

```
INFO:Detectors:
Reentrancy in BSCXNTS._harvest(uint256) (contracts/BSCXNTS.sol#313-341):
        External calls:
        - _transferReferral(_pid,referAmountLv1,referAmountLv2) (contracts/BSCXNTS.sol#328)
                - pool.rewardToken.transfer(referrerLv1,_referAmountLv1.sub(lockAmount)) (contracts/BSCXNTS.sol#352)
                - pool.rewardToken.transfer(referrerLv2,_referAmountLv2.sub(lockAmount_scope_0)) (contracts/BSCXNTS.sol#366)
                - pool.rewardToken.transfer(devaddr,referAmountForDev.sub(lockAmount_scope_1)) (contracts/BSCXNTS.sol#381)
        - pool.rewardToken.transfer(msg.sender,amount.sub(lockAmount)) (contracts/BSCXNTS.sol#332)
        State variables written after the call(s):
        - farmLock(msg.sender,lockAmount,_pid) (contracts/BSCXNTS.sol#333)
                - pool.totalLock = pool.totalLock.add(_amount) (contracts/BSCXNTS.sol#478)
        - farmLock(msg.sender,lockAmount,_pid) (contracts/BSCXNTS.sol#333)
                - totalLocks[pool.rewardToken] = totalLocks[pool.rewardToken].add(_amount) (contracts/BSCXNTS.sol#479)
        - farmLock(msg.sender,lockAmount,_pid) (contracts/BSCXNTS.sol#333)
                - user.lockAmount = user.lockAmount.add(_amount) (contracts/BSCXNTS.sol#477)
                - user.lastUnlockBlock = pool.lockFromBlock (contracts/BSCXNTS.sol#482)
        - user.rewardDebtAtBlock = block.number (contracts/BSCXNTS.sol#334)
        - user.rewardDebt = user.amount.mul(pool.accRewardPerShare).div(1e12) (contracts/BSCXNTS.sol#339)
Reentrancy in BSCXNTS._transferReferral(uint256,uint256,uint256) (contracts/BSCXNTS.sol#343-384):
        External calls:
        - pool.rewardToken.transfer(referrerLv1,_referAmountLv1.sub(lockAmount)) (contracts/BSCXNTS.sol#352)
        State variables written after the call(s):
        - farmLock(referrerLv1,lockAmount,_pid) (contracts/BSCXNTS.sol#353)
                - pool.totalLock = pool.totalLock.add(_amount) (contracts/BSCXNTS.sol#478)
Reentrancy in BSCXNTS._transferReferral(uint256,uint256,uint256) (contracts/BSCXNTS.sol#343-384):
        External calls:
        - pool.rewardToken.transfer(referrerLv1,_referAmountLv1.sub(lockAmount)) (contracts/BSCXNTS.sol#352)
        - pool.rewardToken.transfer(referrerLv2,_referAmountLv2.sub(lockAmount_scope_0)) (contracts/BSCXNTS.sol#366)
        State variables written after the call(s):
        - farmLock(referrerLv2,lockAmount_scope_0,_pid) (contracts/BSCXNTS.sol#367)
                - pool.totalLock = pool.totalLock.add(_amount) (contracts/BSCXNTS.sol#478)
        - farmLock(referrerLv2,lockAmount_scope_0,_pid) (contracts/BSCXNTS.sol#367)
                - totalLocks[pool.rewardToken] = totalLocks[pool.rewardToken].add(_amount) (contracts/BSCXNTS.sol#479)
        - farmLock(referrerLv2,lockAmount_scope_0,_pid) (contracts/BSCXNTS.sol#367)
                - user.lockAmount = user.lockAmount.add(_amount) (contracts/BSCXNTS.sol#477)
                - user.lastUnlockBlock = pool.lockFromBlock (contracts/BSCXNTS.sol#482)
Reentrancy in BSCXNTS._transferReferral(uint256,uint256,uint256) (contracts/BSCXNTS.sol#343-384):
        External calls:
        - pool.rewardToken.transfer(referrerLv1,_referAmountLv1.sub(lockAmount)) (contracts/BSCXNTS.sol#352)
        - pool.rewardToken.transfer(referrerLv2,_referAmountLv2.sub(lockAmount_scope_0)) (contracts/BSCXNTS.sol#366)
        - pool.rewardToken.transfer(devaddr,referAmountForDev.sub(lockAmount_scope_1)) (contracts/BSCXNTS.sol#381)
        State variables written after the call(s):
        - farmLock(devaddr,lockAmount_scope_1,_pid) (contracts/BSCXNTS.sol#382)
                - pool.totalLock = pool.totalLock.add(_amount) (contracts/BSCXNTS.sol#478)
        - farmLock(devaddr,lockAmount_scope_1,_pid) (contracts/BSCXNTS.sol#382)
                - totalLocks[pool.rewardToken] = totalLocks[pool.rewardToken].add(_amount) (contracts/BSCXNTS.sol#479)
        - farmLock(devaddr,lockAmount_scope_1,_pid) (contracts/BSCXNTS.sol#382)
                - user.lockAmount = user.lockAmount.add(_amount) (contracts/BSCXNTS.sol#477)
                - user.lastUnlockBlock = pool.lockFromBlock (contracts/BSCXNTS.sol#482)
Reentrancy in BSCXNTS.claimReward(uint256) (contracts/BSCXNTS.sol#307-311):
        External calls:
        - updatePool(_pid) (contracts/BSCXNTS.sol#309)
                - pool.rewardToken.burn(forBurn) (contracts/BSCXNTS.sol#224)
                - pool.rewardToken.transfer(teamAddresses[_pid],forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#230)
                - pool.rewardToken.transfer(devaddr,forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#233)
        - _harvest(_pid) (contracts/BSCXNTS.sol#310)
                - pool.rewardToken.transfer(referrerLv1,_referAmountLv1.sub(lockAmount)) (contracts/BSCXNTS.sol#352)
                - pool.rewardToken.transfer(msg.sender,amount.sub(lockAmount)) (contracts/BSCXNTS.sol#332)
                - pool.rewardToken.transfer(referrerLv2,_referAmountLv2.sub(lockAmount_scope_0)) (contracts/BSCXNTS.sol#366)
                - pool.rewardToken.transfer(devaddr,referAmountForDev.sub(lockAmount_scope_1)) (contracts/BSCXNTS.sol#381)
        State variables written after the call(s):
        - _harvest(_pid) (contracts/BSCXNTS.sol#310)
                - pool.totalLock = pool.totalLock.add(_amount) (contracts/BSCXNTS.sol#478)
        - _harvest(_pid) (contracts/BSCXNTS.sol#310)
                - totalLocks[pool.rewardToken] = totalLocks[pool.rewardToken].add(_amount) (contracts/BSCXNTS.sol#479)
        - _harvest(_pid) (contracts/BSCXNTS.sol#310)
                - user.lockAmount = user.lockAmount.add(_amount) (contracts/BSCXNTS.sol#477)
                - user.lastUnlockBlock = pool.lockFromBlock (contracts/BSCXNTS.sol#482)
                - user.rewardDebtAtBlock = block.number (contracts/BSCXNTS.sol#334)
                - user.rewardDebt = user.amount.mul(pool.accRewardPerShare).div(1e12) (contracts/BSCXNTS.sol#339)
Reentrancy in BSCXNTS.deposit(uint256,uint256,address) (contracts/BSCXNTS.sol#387-407):
        External calls:
        - updatePool(_pid) (contracts/BSCXNTS.sol#392)
                - pool.rewardToken.burn(forBurn) (contracts/BSCXNTS.sol#224)
                - pool.rewardToken.transfer(teamAddresses[_pid],forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#230)
                - pool.rewardToken.transfer(devaddr,forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#233)
        - _harvest(_pid) (contracts/BSCXNTS.sol#393)
                - pool.rewardToken.transfer(referrerLv1,_referAmountLv1.sub(lockAmount)) (contracts/BSCXNTS.sol#352)
                - pool.rewardToken.transfer(msg.sender,amount.sub(lockAmount)) (contracts/BSCXNTS.sol#332)
                - pool.rewardToken.transfer(referrerLv2,_referAmountLv2.sub(lockAmount_scope_0)) (contracts/BSCXNTS.sol#366)
                - pool.rewardToken.transfer(devaddr,referAmountForDev.sub(lockAmount_scope_1)) (contracts/BSCXNTS.sol#381)
        State variables written after the call(s):
        - _harvest(_pid) (contracts/BSCXNTS.sol#393)
                - pool.totalLock = pool.totalLock.add(_amount) (contracts/BSCXNTS.sol#478)
        - _harvest(_pid) (contracts/BSCXNTS.sol#393)
                - totalLocks[pool.rewardToken] = totalLocks[pool.rewardToken].add(_amount) (contracts/BSCXNTS.sol#479)
        - _harvest(_pid) (contracts/BSCXNTS.sol#393)
                - user.lockAmount = user.lockAmount.add(_amount) (contracts/BSCXNTS.sol#477)
                - user.lastUnlockBlock = pool.lockFromBlock (contracts/BSCXNTS.sol#482)
                - user.rewardDebtAtBlock = block.number (contracts/BSCXNTS.sol#334)
                - user.rewardDebt = user.amount.mul(pool.accRewardPerShare).div(1e12) (contracts/BSCXNTS.sol#339)
```

```
Reentrancy in BSCXNTS.deposit(uint256,uint256,address) (contracts/BSCXNTS.sol#387-407):
        External calls:
        - updatePool(_pid) (contracts/BSCXNTS.sol#392)
                - pool.rewardToken.burn(forBurn) (contracts/BSCXNTS.sol#224)
                - pool.rewardToken.transfer(teamAddresses[_pid],forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#230)
                - pool.rewardToken.transfer(devaddr,forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#233)
        - _harvest(_pid) (contracts/BSCXNTS.sol#393)
                - pool.rewardToken.transfer(referrerLv1,_referAmountLv1.sub(lockAmount)) (contracts/BSCXNTS.sol#352)
                - pool.rewardToken.transfer(msg.sender,amount.sub(lockAmount)) (contracts/BSCXNTS.sol#332)
                - pool.rewardToken.transfer(referrerLv2,_referAmountLv2.sub(lockAmount_scope_0)) (contracts/BSCXNTS.sol#366)
                - pool.rewardToken.transfer(devaddr,referAmountForDev.sub(lockAmount_scope_1)) (contracts/BSCXNTS.sol#381)
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (contracts/BSCXNTS.sol#394)
        State variables written after the call(s):
        - lpBalances[pool.lpToken] = lpBalances[pool.lpToken].add(_amount) (contracts/BSCXNTS.sol#405)
        - referrers[address(msg.sender)] = address(_referrer) (contracts/BSCXNTS.sol#402)
        - user.rewardDebtAtBlock = block.number (contracts/BSCXNTS.sol#396)
        - user.amount = user.amount.add(_amount) (contracts/BSCXNTS.sol#398)
        - user.rewardDebt = user.amount.mul(pool.accRewardPerShare).div(1e12) (contracts/BSCXNTS.sol#399)
Reentrancy in BSCXNTS.emergencyWithdraw(uint256) (contracts/BSCXNTS.sol#429-436):
        External calls:
        - pool.lpToken.safeTransfer(address(msg.sender),user.amount) (contracts/BSCXNTS.sol#432)
        State variables written after the call(s):
        - user.amount = 0 (contracts/BSCXNTS.sol#434)
        - user.rewardDebt = 0 (contracts/BSCXNTS.sol#435)
Reentrancy in BSCXNTS.set(uint256,uint256,bool) (contracts/BSCXNTS.sol#186-194):
        External calls:
        - massUpdatePools() (contracts/BSCXNTS.sol#188)
                - pool.rewardToken.burn(forBurn) (contracts/BSCXNTS.sol#224)
                - pool.rewardToken.transfer(teamAddresses[_pid],forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#230)
                - pool.rewardToken.transfer(devaddr,forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#233)
        State variables written after the call(s):
        - pool.allocPoint = _allocPoint (contracts/BSCXNTS.sol#193)
        - totalAllocPoints[pool.rewardToken] = totalAllocPoints[pool.rewardToken].sub(pool.allocPoint).add(_allocPoint) (contracts/BSCXNTS.sol#192)
Reentrancy in BSCXNTS.unlock(uint256) (contracts/BSCXNTS.sol#504-519):
        External calls:
        - pool.rewardToken.transfer(msg.sender,amount) (contracts/BSCXNTS.sol#514)
        State variables written after the call(s):
        - pool.totalLock = pool.totalLock.sub(amount) (contracts/BSCXNTS.sol#517)
        - user.lockAmount = user.lockAmount.sub(amount) (contracts/BSCXNTS.sol#515)
        - user.lastUnlockBlock = block.number (contracts/BSCXNTS.sol#516)
```

```
Reentrancy in BSCXNTS.updatePool(uint256) (contracts/BSCXNTS.sol#205-239):
        External calls:
        - pool.rewardToken.burn(forBurn) (contracts/BSCXNTS.sol#224)
        - pool.rewardToken.transfer(teamAddresses[_pid],forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#230)
        State variables written after the call(s):
        - farmLock(teamAddresses[_pid],lockAmount,_pid) (contracts/BSCXNTS.sol#231)
                - pool.totalLock = pool.totalLock.add(_amount) (contracts/BSCXNTS.sol#478)
        - farmLock(teamAddresses[_pid],lockAmount,_pid) (contracts/BSCXNTS.sol#231)
                - totalLocks[pool.rewardToken] = totalLocks[pool.rewardToken].add(_amount) (contracts/BSCXNTS.sol#479)
Reentrancy in BSCXNTS.updatePool(uint256) (contracts/BSCXNTS.sol#205-239):
        External calls:
        - pool.rewardToken.burn(forBurn) (contracts/BSCXNTS.sol#224)
        - pool.rewardToken.transfer(devaddr,forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#233)
        State variables written after the call(s):
        - farmLock(devaddr,lockAmount,_pid) (contracts/BSCXNTS.sol#234)
                - pool.totalLock = pool.totalLock.add(_amount) (contracts/BSCXNTS.sol#478)
        - farmLock(devaddr,lockAmount,_pid) (contracts/BSCXNTS.sol#234)
                - totalLocks[pool.rewardToken] = totalLocks[pool.rewardToken].add(_amount) (contracts/BSCXNTS.sol#479)
Reentrancy in BSCXNTS.updatePool(uint256) (contracts/BSCXNTS.sol#205-239):
        External calls:
        - pool.rewardToken.burn(forBurn) (contracts/BSCXNTS.sol#224)
        - pool.rewardToken.transfer(teamAddresses[_pid],forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#230)
        - pool.rewardToken.transfer(devaddr,forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#233)
        State variables written after the call(s):
        - pool.accRewardPerShare = pool.accRewardPerShare.add(forFarmer.mul(1e12).div(lpSupply)) (contracts/BSCXNTS.sol#237)
        - pool.lastRewardBlock = block.number (contracts/BSCXNTS.sol#238)
Reentrancy in BSCXNTS.withdraw(uint256,uint256) (contracts/BSCXNTS.sol#410-426):
        External calls:
        - updatePool(_pid) (contracts/BSCXNTS.sol#416)
                - pool.rewardToken.burn(forBurn) (contracts/BSCXNTS.sol#224)
                - pool.rewardToken.transfer(teamAddresses[_pid],forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#230)
                - pool.rewardToken.transfer(devaddr,forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#233)
        - _harvest(_pid) (contracts/BSCXNTS.sol#417)
                - pool.rewardToken.transfer(referrerLv1,_referAmountLv1.sub(lockAmount)) (contracts/BSCXNTS.sol#352)
                - pool.rewardToken.transfer(msg.sender,amount.sub(lockAmount)) (contracts/BSCXNTS.sol#332)
                - pool.rewardToken.transfer(referrerLv2,_referAmountLv2.sub(lockAmount_scope_0)) (contracts/BSCXNTS.sol#366)
                - pool.rewardToken.transfer(devaddr,referAmountForDev.sub(lockAmount_scope_1)) (contracts/BSCXNTS.sol#381)
        State variables written after the call(s):
        - _harvest(_pid) (contracts/BSCXNTS.sol#417)
                - pool.totalLock = pool.totalLock.add(_amount) (contracts/BSCXNTS.sol#478)
        - _harvest(_pid) (contracts/BSCXNTS.sol#417)
                - totalLocks[pool.rewardToken] = totalLocks[pool.rewardToken].add(_amount) (contracts/BSCXNTS.sol#479)
        - _harvest(_pid) (contracts/BSCXNTS.sol#417)
                - user.lockAmount = user.lockAmount.add(_amount) (contracts/BSCXNTS.sol#477)
                - user.lastUnlockBlock = pool.lockFromBlock (contracts/BSCXNTS.sol#482)
                - user.rewardDebtAtBlock = block.number (contracts/BSCXNTS.sol#334)
                - user.rewardDebt = user.amount.mul(pool.accRewardPerShare).div(1e12) (contracts/BSCXNTS.sol#339)
        - user.amount = user.amount.sub(_amount) (contracts/BSCXNTS.sol#420)
```

Add nonReentrant modifier to close all possibilities of Re-entrancy in future as well.

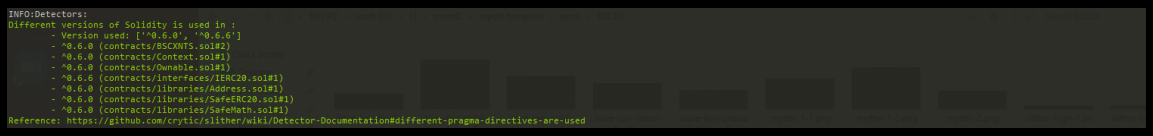
```
INFO:Detectors:
BSCXNTS.updatePool(uint256) (contracts/BSCXNTS.sol#205-239) ignores return value by pool.rewardToken.burn(forBurn) (contracts/BSCXNTS.sol#224)
BSCXNTS.updatePool(uint256) (contracts/BSCXNTS.sol#205-239) ignores return value by pool.rewardToken.transfer(teamAddresses[_pid],forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#230)
BSCXNTS.updatePool(uint256) (contracts/BSCXNTS.sol#205-239) ignores return value by pool.rewardToken.transfer(devaddr,forDev.sub(lockAmount)) (contracts/BSCXNTS.sol#233)
BSCXNTS._harvest(uint256) (contracts/BSCXNTS.sol#313-341) ignores return value by pool.rewardToken.transfer(msg.sender,amount.sub(lockAmount)) (contracts/BSCXNTS.sol#332)
BSCXNTS._transferReferral(uint256,uint256,uint256) (contracts/BSCXNTS.sol#343-384) ignores return value by pool.rewardToken.transfer(referrerLv1,_referAmountLv1.sub(lockAmount)) (contracts/BSCXNTS.sol#352)
BSCXNTS._transferReferral(uint256,uint256,uint256) (contracts/BSCXNTS.sol#343-384) ignores return value by pool.rewardToken.transfer(referrerLv2,_referAmountLv2.sub(lockAmount_scope_0)) (contracts/BSCXNTS.sol#366)
BSCXNTS._transferReferral(uint256,uint256,uint256) (contracts/BSCXNTS.sol#343-384) ignores return value by pool.rewardToken.transfer(devaddr,referAmountForDev.sub(lockAmount_scope_1)) (contracts/BSCXNTS.sol#381)
BSCXNTS.unlock(uint256) (contracts/BSCXNTS.sol#504-519) ignores return value by pool.rewardToken.transfer(msg.sender,amount) (contracts/BSCXNTS.sol#514)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

Issue already mentioned above regarding ignoring return values.

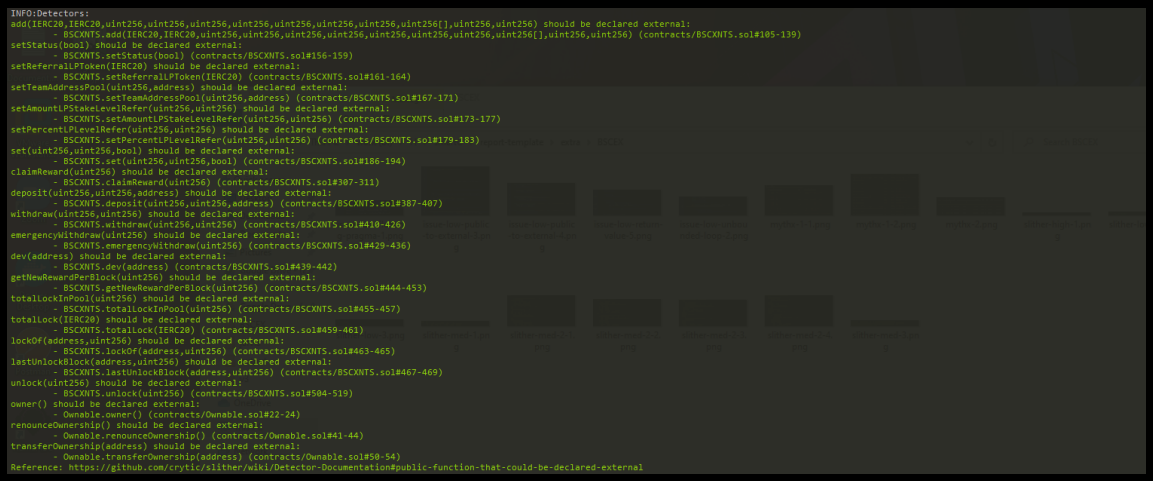
```
INFO:Detectors:
BSCXNTS.constructor(address,uint256,uint256,uint256,uint256,IERC20)._devaddr (contracts/BSCXNTS.sol#83) lacks a zero-check on :
        - devaddr = _devaddr (contracts/BSCXNTS.sol#90)
BSCXNTS.dev(address)._devaddr (contracts/BSCXNTS.sol#439) lacks a zero-check on :
        - devaddr = _devaddr (contracts/BSCXNTS.sol#441)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

Valid issue, Kindly add zero-address validation at Line #90 and Line #441 for _devaddr since it is consuming address values directly from user.

Add nonReentrant modifier to close all possibilities of Re-entrancy in future as well.



Use same pragma throughout the contracts and libraries. Also it is better to remove floating pragma.



Issue already mentioned above regarding function could have marked as external.

3.7 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. In addition, security detections are only in scope.

Results:

BSCXNTS.sol

Report for BSCXNTS.sol
https://dashboard.mythx.io/#/console/analyses/8a875871-0f51-403b-9905-17de73c4a456

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
105	(SWC-000) Unknown	Medium	Function could be marked as external.
126	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
146	(SWC-128) DoS With Block Gas Limit	Medium	Implicit loop over unbounded data structure.
156	(SWC-000) Unknown	Medium	Function could be marked as external.
161	(SWC-000) Unknown	Medium	Function could be marked as external.
167	(SWC-000) Unknown	Medium	Function could be marked as external.
173	(SWC-000) Unknown	Medium	Function could be marked as external.
179	(SWC-000) Unknown	Medium	Function could be marked as external.
186	(SWC-000) Unknown	Medium	Function could be marked as external.
199	(SWC-128) DoS With Block Gas Limit	Medium	Loop over unbounded data structure.
207	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
214	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
238	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

273	(SWC-128) DoS With Block Gas Limit	Medium	Implicit loop over unbounded data structure.
273	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
298	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
307	(SWC-000) Unknown	Medium	Function could be marked as external.
334	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
387	(SWC-000) Unknown	Medium	Function could be marked as external.
396	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
410	(SWC-000) Unknown	Medium	Function could be marked as external.
429	(SWC-000) Unknown	Medium	Function could be marked as external.
439	(SWC-000) Unknown	Medium	Function could be marked as external.
444	(SWC-000) Unknown	Medium	Function could be marked as external.
447	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
447	(SWC-128) DoS With Block Gas Limit	Low	Implicit loop over unbounded data structure.
455	(SWC-000) Unknown	Medium	Function could be marked as external.
459	(SWC-000) Unknown	Medium	Function could be marked as external.
463	(SWC-000) Unknown	Medium	Function could be marked as external.
467	(SWC-000) Unknown	Medium	Function could be marked as external.
491	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
494	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
498	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.
504	(SWC-000) Unknown	Medium	Function could be marked as external.
516	(SWC-120) Weak Sources of Randomness from Chain Attributes	Low	Potential use of "block.number" as source of randonmness.

Ownable.sol

Report for Ownable.sol
https://dashboard.mythx.io/#/console/analyses/8a875871-0f51-403b-9905-17de73c4a456

Line	SWC Title	Severity	Short Description
22	(SWC-000) Unknown	Medium	Function could be marked as external.
41	(SWC-000) Unknown	Medium	Function could be marked as external.
50	(SWC-000) Unknown	Medium	Function could be marked as external.

THANK YOU FOR CHOOSING

// HALBORN