QuillAudits

Audit Report
December, 2020

Contents

Introduction

This Audit Report mainly focuses on the overall security of COX Smart Contract. With this report, we have tried to ensure the reliability and correctness of their smart contract by a complete and rigorous assessment of their system's architecture and the smart contract codebase.

Auditing Approach and Methodologies applied

The Quillhash team has performed rigorous testing of the project starting with analysing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third party smart contracts and libraries.

Our team then performed a formal line by line inspection of the Smart Contract to find any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In the Unit testing Phase, we coded/conducted custom unit tests written for each function in the contract to verify that each function works as expected.

In Automated Testing, We tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was tested in collaboration of our multiple team members and this included -

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the whole process.
- Analysing the complexity of the code in-depth and detailed, manual review of the code, line-by-line.
- Deploying the code on testnet using multiple clients to run live tests.
- Analysing failure preparations to check how the Smart Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analysing the security of the on-chain data.

Audit Details

Project Name: **COXENA**

Languages: Solidity (Smart contract)

Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Securify, Mythril, Contract Library, Slither, SmartCheck

Audit Goals

The focus of the audit was to verify that the Smart Contract System is secure, resilient and working according to the specifications. The audit activities can be grouped in the following three categories:

Security
Identifying security related issues within each contract and the system of contract.

Sound Architecture
Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality
A full review of the contract source code. The primary areas of focus include:

- Accuracy
- Readability
- Sections of code with high complexity
- Quantity and quality of test coverage

Issue Categories

Every issue in this report was assigned a severity level from the following:

High level severity issues

Issues on this level are critical to the smart contract's performance/ functionality and should be fixed before moving to a live environment.

Medium level severity issues

Issues on this level could potentially bring problems and should eventually be fixed.

Low level severity issues

Issues on this level are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Number of issues per severity

	Low	Medium	High	Recommendations
Open	0	0	0	13
Closed	2	4	2	5

Manual Audit

For this section, the code was tested/read line by line by our developers. We also used Remix IDE's JavaScript VM and Kovan networks to test the contract functionality. The verified contract on Kovan used for testing: https://kovan.etherscan.io/address/0x4c9A8CcfB7C64A1C776696c11130e5EBa5DE73A2

Low level severity issues

1. **Status: Fixed and Closed**

   The pragma versions used within the contract are not locked. Consider using version between 0.6.0 to 0.6.12 for deploying the contracts as it does not compile for any other version and can be confusing for a developer. Solidity source files indicate the versions of the compiler they can be compiled with.

   pragma solidity <=0.7.0; // bad: compiles with 0.7.0 and below
   pragma solidity 0.6.12; // good : compiles w 0.6.12 only

2. **Status: Fixed and Closed**

   A typographical error was noticed, change "olny" to "only".

```
modifier onlyPresaleContract {
    require(msg.sender == address(presale), "olny presale can call this.");
    _;
}
```

Medium level severity issues

1. **Status: Fixed and Closed**

   transfer function can be converted to external to save gas for users. The only place where it is used in the contract is the changeAdmin function, where the internal _transfer function can be called. So line number 700 can be changed to:
   _transfer(msg.sender, _to,_board.parties[msg.sender].balance);

2. **Status: Fixed and Closed**

   The following functions should be declared external as they are not used anywhere else in the contract. This saves gas on contract deployment.
   - cycleUnstakeStatusOf
   - CheckStakeDetails
   - viewTranferDetails
   - minStake
   - stakeStats
   - viewEnableStakeBonus
   - ViewBurnLimit
   - ViewStartStakeBonusDate
   -ViewEntry_Days
   - ViewStake_Bonus_Interval

3. **Status: Fixed and Closed**

   The function **setEnableTax** will run successfully even though the enableTax parameter provided is any value other than 0 and 1. In that case no state is changed and unnecessary gas is lost. We advise to add a “require” message that checks if the value is 0 or 1 in the beginning of this function.

```
function setEnableTax(uint enableTax) external virtual onlyOwner returns (bool) {
    if(enableTax == 1) _enableTax = true;
    else if(enableTax == 0) _enableTax = false;
    return _enableTax;
}
```

There is the same issue as above for the following functions:

- **setEnableStakeBonus**
- **setEnableStake**

4. **Status: Fixed and Closed**

It is recommended to perform multiplication before division. Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

```
929  function poolProfitOf(address account,uint256 cycle) public view returns (uint256) {
930      if(_board.parties[account].monthlyUnStake[cycle] == true) return 0;
931      uint256 userStaked = cycleStakePoolOf(account,cycle);
932      if(userStaked <= 0) return 0;
933
934      uint256 cycletotalStaked = cycleStakePoolTotal(cycle);
935      uint256 share = userStaked.div(cycletotalStaked).mul(_board.totalMonthlyStakePoolReward[cycle]);
936      return uint256(share.div(_Scale));
937  }
```

Here in line 935, we recommend multiplication is done before division.

High level severity issues

1. **Status: Fixed and Closed**

The changeAdmin function has a wrong implementation for changing the admin.

```
697      function changeAdmin(address _to) external virtual{
698          require(msg.sender == _board.owner);
699
700          transfer(_to,_board.parties[msg.sender].balance);
701          eliters(_to,1);
702
703          _board.owner = msg.sender;
704
705      }
```

On line number 703, **_board.owner = msg.sender** is a wrong statement as per our analysis. It should be replaced with **_board.owner = _to** as we are changing the admin to "_to" address by transferring all the balances to that address. Although, if the current logic is desired then the assignment in line 703 is useless as the "require" message in line 698 makes sure that _board.owner is already the msg.sender.

2. **Status: Fixed and Closed**

   The ReentrancyGuard contract is not useful as there are no possible scenarios where the nonReentrant modifier will be useful. NonReentrancy is usually helpful in case of functions that interact with other contracts. Remove it will reduce significant code and resolve the following issues as well:

   - The contract ReentrancyGuard already inherits the contract Initializable. So the COX contract does not need to inherit both ReentrancyGuard and Initializable. Just inheriting ReentrancyGuard is enough for the COX contract.

   - State variable **_____gap** is declared duplicately in contract ReentrancyGuard as it is already declared in Initializable contract. This can be avoided as it is a case of state variable shadowing.

Recommendations

1. **Status: Open**

   Add events for most state changes. For example, after modifying balances for parties in _transfer function when adding rewards. Events should be fired with all state variable updates as good practice.

2. **Status: Open**

   To handle balance updates in the contract efficiently with events, we advise adding a public mint function that adds balances for any party along with the total supply if needed.

3. **Status: Open**

   Follow solidity style guide for better readability:
   https://docs.soliditylang.org/en/v0.7.5/style-guide.html

For example, Functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)
- external
- public
- internal
- private
- Within a grouping, place the view and pure functions last.

Note: Linting violations can be easily fixed using linters like solhint.

4. **Status: Open**

   Add Natspecs comments to all functions for better understanding regarding what the parameters mean and what the function does.

5. **Status: Open**

   All the “require” statements used in the contract should also specify error messages for easy debugging.

6. **Status: Open**

   Modify onlyOwner modifier to onlyBoardOwner for better understanding.

7. **Status: Fixed and Closed**

   The following functions are missing a zero check for addresses in parameters. We recommend that they are added:

   - Address **party** in function **eliters (Closed)**
   - Address **bonusPool** in function **dEsad (Closed with explanation: only owner functions can skip address zero checks)**
   - Addresses **bonusPool** and **_preSale** not checked for zero addresses in **initialize** function **(Closed with explanation: contract deployer is a smart user and hence can skip zero check)**

8. **Status: Fixed and Closed**

   Explicitly state visibility of **SECONDS_PER_DAY**, right now it is defaulted to internal. This can help avoid ambiguity in contract code. We recommend using “private” modifier here.

```
address private _bonusPool;
uint256 constant SECONDS_PER_DAY = 24 * 60 * 60;
```

9. **Status: Fixed and Closed**

   We recommend adding a getter function for getting the owner of the contract for tracking the _board.owner. Currently there is no way of knowing which address is the owner of the contract which can create confusion.

10. **Status: Fixed and Closed**

    We recommend avoiding comparing variables to a boolean constant. This will save gas costs.
    Examples:
    ” if(_enableStake_Bonus == true)” can be replaced with
    “if(_enableStake_Bonus)”

    “if(_board.totalSupply < _Burnt_Limit || _board.parties[sender].elite || _enableTax == false)”
    can be replaced with
    “if(_board.totalSupply < _Burnt_Limit || _board.parties[sender].elite || !_enableTax)”

11. **Status: Open**

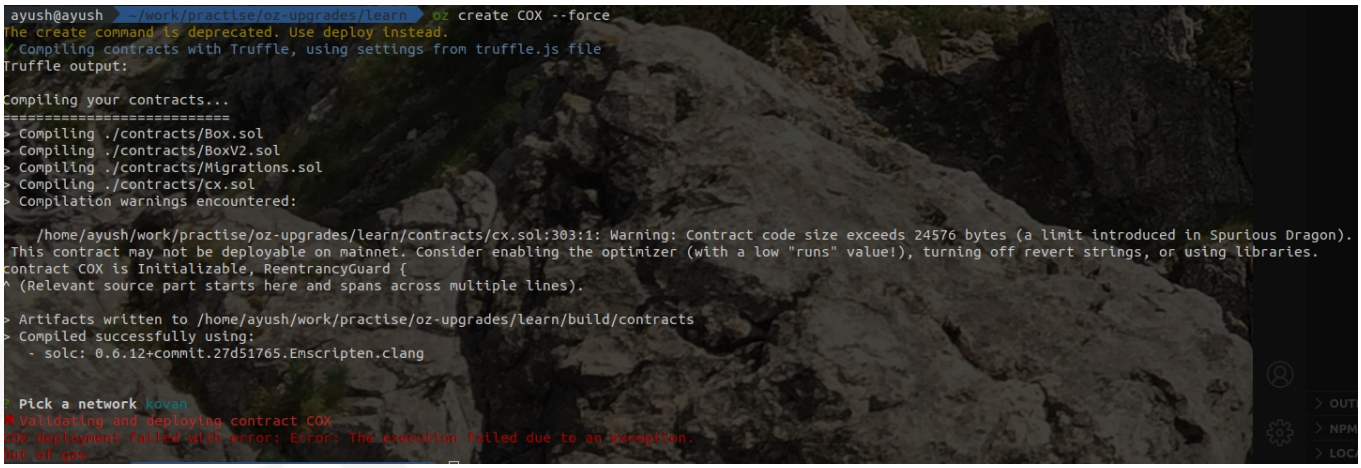
    We recommend following the solidity naming conventions. There are a lot of naming in the contract that deviate from the conventions. For example, _enableStake_Bonus does not follow mixed case format.

12. Status: Fixed and Closed

The variable **“address private presale”** does not have a setter function and can be only set in the initialize function. So in case there is some typo error for the _presale parameter in the initialize function and the wrong address is provided by mistake, it can’t be modified later. So to deal with this, we advise to create an onlyOwner setter function which provides the flexibility to change this variable after the contract is deployed.

13. Status: Open

The contract follows OpenZeppelin initialize pattern for upgradable contracts but OZ CLI was unable to deploy it on Kovan and returns the following exception:

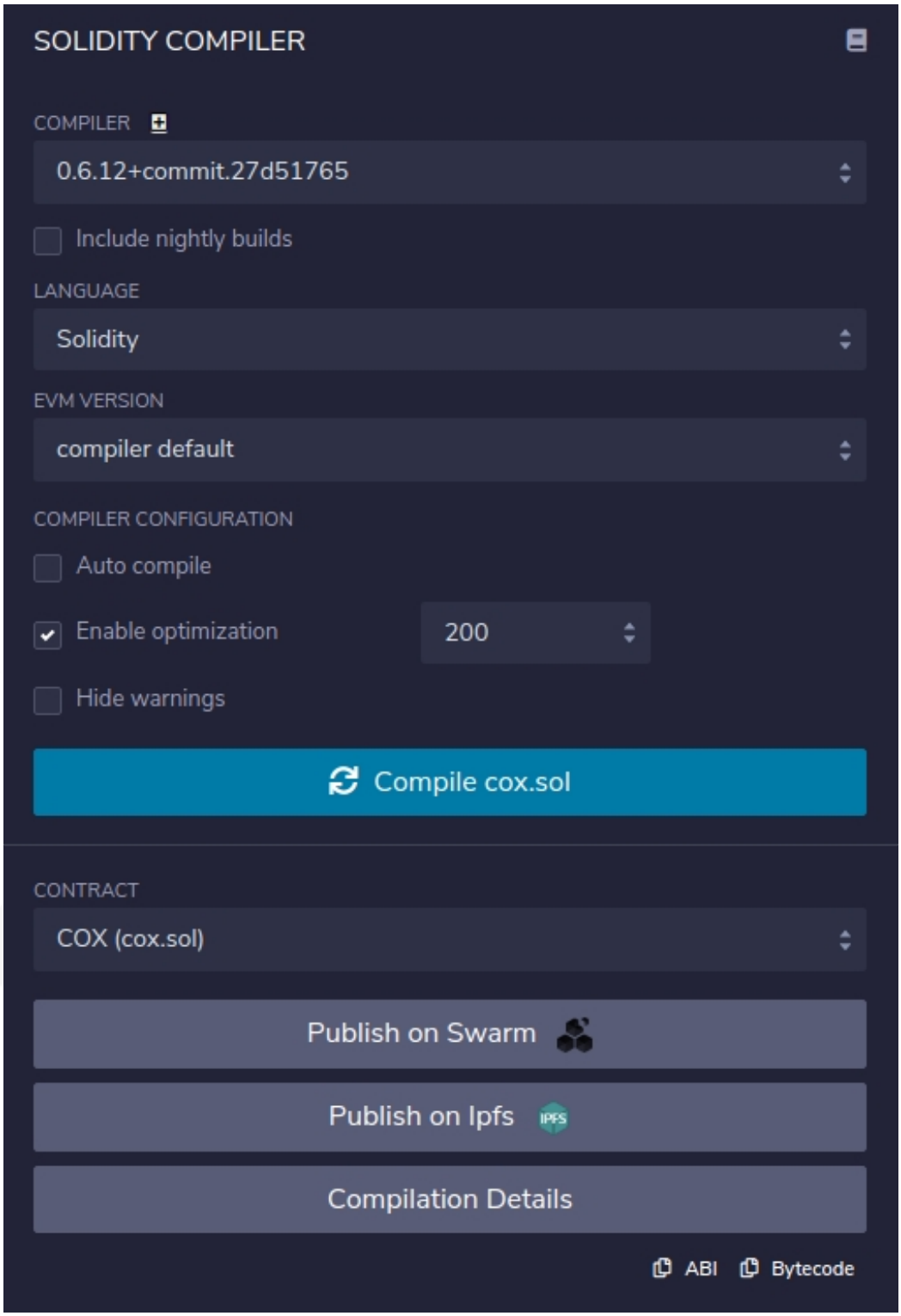


This is because the contract code size exceeds 24576 bytes. It is recommended to reduce code size by removing unused code as stated earlier as well as consider implementing libraries and turning off revert strings to decrease code size. This recommendation is necessary only if the client expects to deploy an upgradable contract. Other considerable recommendations can be found here.

Automated Audit

Remix Compiler Warnings

No errors were thrown by the compiler:



Contract Library

Contract-library contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analysis applied to these in real time.

We performed analysis using contract Library on the mainnet address of the COX contract: 0x4c9a8ccfb7c64a1c776696c11130e5eba5de73a2

Analysis summary can be accessed here:
https://contract-library.com/contracts/Kovan/0x4c9a8ccfb7c64a1c776696c11130e5eba5de73a2

It did not return any issue during the analysis.

SmartCheck

Smartcheck is a tool for automated static analysis of Solidity source code for security vulnerabilities and best practices. SmartCheck translates Solidity source code into an XML-based intermediate representation and checks it against XPath patterns. Smartcheck shows significant improvements over existing alternatives in terms of false discovery rate (FDR) and false negative rate (FNR). It gave the following result for the COX contract:
https://tool.smartdec.net/scan/0dd60445d34447598f8fadd37d99943f

SmartCheck did not detect any high severity issue. All the considerable issues raised by SmartCheck are already covered in the Manual Audit section of this report.

Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has the critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

```
➜ COX git:(master) ✗ slither .
'npx truffle@5.1.39 compile --all' running (use --truffle-version truffle@x.x.x to use specific version)

INFO:Detectors:
ReentrancyGuard._______gap (CoxContract_flat.sol#101) shadows:
        - Initializable._______gap (CoxContract_flat.sol#62)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
COX._transfer(address,address,uint256) (CoxContract_flat.sol#584-645) performs a multiplication on the result of a division:
        -toDistribute = amount.mul(_transferRewardRate).div(100) (CoxContract_flat.sol#604)
        -toDistributeForStakers = toDistribute.mul(_transferTaxStakers).div(_transferRatioTotal) (CoxContract_flat.sol#609)
COX._transfer(address,address,uint256) (CoxContract_flat.sol#584-645) performs a multiplication on the result of a division:
        -toDistributeForStakers = toDistribute.mul(_transferTaxStakers).div(_transferRatioTotal) (CoxContract_flat.sol#609)
        -_board.retPerStakeToken = _board.retPerStakeToken.add(toDistributeForStakers.mul(_Scale).div(_board.totalStaked))
(CoxContract_flat.sol#615)
COX._transfer(address,address,uint256) (CoxContract_flat.sol#584-645) performs a multiplication on the result of a division:
        -toDistribute = amount.mul(_transferRewardRate).div(100) (CoxContract_flat.sol#604)
        -toDistributeForStakePoolProvider = toDistribute.mul(_transferTaxLongHolder).div(_transferRatioTotal)
(CoxContract_flat.sol#622)
COX._transfer(address,address,uint256) (CoxContract_flat.sol#584-645) performs a multiplication on the result of a division:
        -toDistributeForStakePoolProvider = toDistribute.mul(_transferTaxLongHolder).div(_transferRatioTotal)
(CoxContract_flat.sol#622)
        -_board.totalMonthlyStakePoolReward[currentCycle] =
_board.totalMonthlyStakePoolReward[currentCycle].add(toDistributeForStakePoolProvider.mul(_Scale)) (CoxContract_flat.sol#624)
COX.unStake(uint256) (CoxContract_flat.sol#795-846) performs a multiplication on the result of a division:
        -reward = amount.mul(_unstakeRewardRate).div(100) (CoxContract_flat.sol#799)
        -toStakers = reward.mul(_unstakeTaxStaker).div(_unstakeRatioTotal) (CoxContract_flat.sol#805)
COX.unStake(uint256) (CoxContract_flat.sol#795-846) performs a multiplication on the result of a division:
        -reward = amount.mul(_unstakeRewardRate).div(100) (CoxContract_flat.sol#799)
        -toStakePoolProviders = reward.mul(_unstakeTaxLongHolders).div(_unstakeRatioTotal) (CoxContract_flat.sol#806)
COX.unStake(uint256) (CoxContract_flat.sol#795-846) performs a multiplication on the result of a division:
        -toStakers = reward.mul(_unstakeTaxStaker).div(_unstakeRatioTotal) (CoxContract_flat.sol#805)
        -_board.retPerStakeToken = _board.retPerStakeToken.add(toStakers.mul(_Scale).div(_board.totalStaked))
(CoxContract_flat.sol#824)
COX.unStake(uint256) (CoxContract_flat.sol#795-846) performs a multiplication on the result of a division:
        -toStakePoolProviders = reward.mul(_unstakeTaxLongHolders).div(_unstakeRatioTotal) (CoxContract_flat.sol#806)
        -_board.totalMonthlyStakePoolReward[currentCycle] =
_board.totalMonthlyStakePoolReward[currentCycle].add(toStakePoolProviders.mul(_Scale)) (CoxContract_flat.sol#827)
COX.poolProfitOf(address,uint256) (CoxContract_flat.sol#929-937) performs a multiplication on the result of a division:
        -share = userStaked.div(cycletotalStaked).mul(_board.totalMonthlyStakePoolReward[cycle]) (CoxContract_flat.sol#935)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
COX.initialize(string,string,uint256,uint256,address,uint256,address).name (CoxContract_flat.sol#390) shadows:
        - COX.name() (CoxContract_flat.sol#440-442) (function)
COX.initialize(string,string,uint256,uint256,address,uint256,address).symbol (CoxContract_flat.sol#390) shadows:
        - COX.symbol() (CoxContract_flat.sol#448-450) (function)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
COX.setTransferDetails(uint256,uint256,uint256,uint256,uint256,uint256) (CoxContract_flat.sol#891-900) should emit an event for:
        - _transferTaxRate = transferTaxRate (CoxContract_flat.sol#893)
        - _transferRewardRate = transferRewardRate (CoxContract_flat.sol#894)
        - _transferBurnRate = transferBurnRate (CoxContract_flat.sol#895)
        - _transferTaxStakers = transferTaxStakers (CoxContract_flat.sol#896)
        - _transferTaxLongHolder = transferTaxLongHolder (CoxContract_flat.sol#897)
        - _transferTaxReferral = transferTaxReferral (CoxContract_flat.sol#898)
        - _transferRatioTotal = transferTaxLongHolder.add(transferTaxStakers) (CoxContract_flat.sol#899)
COX.setMinStake(uint256) (CoxContract_flat.sol#965-969) should emit an event for:
        - _Min_Stake = amount (CoxContract_flat.sol#967)
COX.setBurnLimit(uint256) (CoxContract_flat.sol#1064-1067) should emit an event for:
        - _Burnt_Limit = limit (CoxContract_flat.sol#1065)
COX.setStartStakeBonusDate(uint256) (CoxContract_flat.sol#1074-1077) should emit an event for:
        - _StartStakeBonusDate = StartStakeBonusDate (CoxContract_flat.sol#1075)
COX.setEntryDays(uint256) (CoxContract_flat.sol#1083-1086) should emit an event for:
        - _Entry_Days = Entry_Days (CoxContract_flat.sol#1084)
COX.setStake_Bonus_Interval(uint256) (CoxContract_flat.sol#1092-1095) should emit an event for:
        - _Stake_Bonus_Interval = Stake_Bonus_Interval (CoxContract_flat.sol#1093)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
COX.initialize(string,string,uint256,uint256,address,uint256,address).bonusPool (CoxContract_flat.sol#390) lacks a zero-check on :
                - _bonusPool = bonusPool (CoxContract_flat.sol#397)
COX.initialize(string,string,uint256,uint256,address,uint256,address)._presale (CoxContract_flat.sol#390) lacks a zero-check on :
                - presale = _presale (CoxContract_flat.sol#426)
COX.dEsad(address).bonusPool (CoxContract_flat.sol#1024) lacks a zero-check on :
                - _bonusPool = bonusPool (CoxContract_flat.sol#1025)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
COX._transfer(address,address,uint256) (CoxContract_flat.sol#584-645) uses timestamp for comparisons
        Dangerous comparisons:
        - currentCycle > 0 && _enableStake_Bonus == true (CoxContract_flat.sol#623)
        - _enableStake_Bonus == true && currentCycle <= 0 (CoxContract_flat.sol#625)
COX.unStake(uint256) (CoxContract_flat.sol#795-846) uses timestamp for comparisons
        Dangerous comparisons:
        - currentCycle <= 0 (CoxContract_flat.sol#809)
COX.SubmitStakePool() (CoxContract_flat.sol#914-927) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(currentCycle > 0,Start pool reward has not started) (CoxContract_flat.sol#919)
        - require(bool,string)(_board.parties[msg.sender].monthlyStakePool[currentCycle] == 0,You already submit stake)
(CoxContract_flat.sol#920)
        - require(bool,string)(getDayOFCurrentCycle() <= _Entry_Days,You can not submit stake pool till next cycle)
(CoxContract_flat.sol#921)
COX.redeemStakePoolProfit(uint256) (CoxContract_flat.sol#939-949) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(getCurrentCycle() > cycle,can not redeem profit for the current cycle) (CoxContract_flat.sol#940)
COX.reStakePoolProfit(uint256) (CoxContract_flat.sol#951-963) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(getCurrentCycle() > cycle,can not redeem profit for the current cycle) (CoxContract_flat.sol#952)
COX.getCurrentCycle() (CoxContract_flat.sol#983-986) uses timestamp for comparisons
        Dangerous comparisons:
        - now <= _StartStakeBonusDate (CoxContract_flat.sol#984)
COX.getDayOFCurrentCycle() (CoxContract_flat.sol#988-991) uses timestamp for comparisons
        Dangerous comparisons:
        - now <= _StartStakeBonusDate (CoxContract_flat.sol#989)
COX._diffDays(uint256,uint256) (CoxContract_flat.sol#1028-1031) uses timestamp for comparisons
```

```
Dangerous comparisons:
	- require(bool)(fromTimestamp <= toTimestamp) (CoxContract_flat.sol#1029)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Initializable.isConstructor() (CoxContract_flat.sol#49-59) uses assembly
	- INLINE ASM (CoxContract_flat.sol#57)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
COX._transfer(address,address,uint256) (CoxContract_flat.sol#584-645) compares to a boolean constant:
	-_enableStake_Bonus == true (CoxContract_flat.sol#608)
COX._transfer(address,address,uint256) (CoxContract_flat.sol#584-645) compares to a boolean constant:
	-_board.totalStaked > 0 && _enableStake == true (CoxContract_flat.sol#614)
COX._transfer(address,address,uint256) (CoxContract_flat.sol#584-645) compares to a boolean constant:
	-currentCycle > 0 && _enableStake_Bonus == true (CoxContract_flat.sol#623)
COX._transfer(address,address,uint256) (CoxContract_flat.sol#584-645) compares to a boolean constant:
	-_enableStake_Bonus == true && currentCycle <= 0 (CoxContract_flat.sol#625)
COX._transfer(address,address,uint256) (CoxContract_flat.sol#584-645) compares to a boolean constant:
	-_board.totalSupply < _Burnt_Limit || _board.parties[sender].elite || _enableTax == false (CoxContract_flat.sol#593)
COX.unStake(uint256) (CoxContract_flat.sol#795-846) compares to a boolean constant:
	-_enableStake_Bonus == true (CoxContract_flat.sol#804)
COX.poolProfitOf(address,uint256) (CoxContract_flat.sol#929-937) compares to a boolean constant:
	-_board.parties[account].monthlyUnStake[cycle] == true (CoxContract_flat.sol#930)
COX.redeemStakePoolProfit(uint256) (CoxContract_flat.sol#939-949) compares to a boolean constant:
	-require(bool,string)(_board.parties[msg.sender].monthlyUnStake[cycle] != true,you have unstake record this cycle)
(CoxContract_flat.sol#941)
COX.reStakePoolProfit(uint256) (CoxContract_flat.sol#951-963) compares to a boolean constant:
	-require(bool,string)(_board.parties[msg.sender].monthlyUnStake[cycle] != true,you have unstake record this cycle)
(CoxContract_flat.sol#953)
COX._register(address) (CoxContract_flat.sol#994-1007) compares to a boolean constant:
	-_board.parties[msg.sender].notNew == true (CoxContract_flat.sol#995)
COX.registerPresale(address,address) (CoxContract_flat.sol#1009-1022) compares to a boolean constant:
	-_board.parties[account].notNew == true (CoxContract_flat.sol#1010)
COX.whenStakeEnabled() (CoxContract_flat.sol#365-368) compares to a boolean constant:
	-require(bool,string)(_enableStake == true,Can only be called when Staking is Enabled.) (CoxContract_flat.sol#366)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Pragma version<=0.7.0 (CoxContract_flat.sol#3) uses lesser than
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable Initializable._______gap (CoxContract_flat.sol#62) is not in mixedCase
Variable ReentrancyGuard._______gap (CoxContract_flat.sol#101) is not in mixedCase
Parameter COX.initialize(string,string,uint256,uint256,address,uint256,address).StartStakeBonusDate (CoxContract_flat.sol#390) is
not in mixedCase
Parameter COX.initialize(string,string,uint256,uint256,address,uint256,address).Entry_Days (CoxContract_flat.sol#390) is not in
mixedCase
Parameter COX.initialize(string,string,uint256,uint256,address,uint256,address).Stake_Bonus_Interval (CoxContract_flat.sol#390) is
not in mixedCase
Parameter COX.initialize(string,string,uint256,uint256,address,uint256,address)._presale (CoxContract_flat.sol#390) is not in
mixedCase
Parameter COX.changeAdmin(address)._to (CoxContract_flat.sol#697) is not in mixedCase
Parameter COX.eliters(address,uint256)._status (CoxContract_flat.sol#707) is not in mixedCase
Function COX.CheckStakeDetails() (CoxContract_flat.sol#882-888) is not in mixedCase
Function COX.SubmitStakePool() (CoxContract_flat.sol#914-927) is not in mixedCase
Parameter COX.registerPresale(address,address)._referral (CoxContract_flat.sol#1009) is not in mixedCase
Parameter COX.setEnableStakeBonus(uint256).enableStake_Bonus (CoxContract_flat.sol#1054) is not in mixedCase
Function COX.ViewBurnLimit() (CoxContract_flat.sol#1069-1071) is not in mixedCase
Parameter COX.setStartStakeBonusDate(uint256).StartStakeBonusDate (CoxContract_flat.sol#1074) is not in mixedCase
```

```
Function COX.ViewStartStakeBonusDate() (CoxContract_flat.sol#1079-1081) is not in mixedCase
Parameter COX.setEntryDays(uint256).Entry_Days (CoxContract_flat.sol#1083) is not in mixedCase
Function COX.ViewEntry_Days() (CoxContract_flat.sol#1088-1090) is not in mixedCase
Function COX.setStake_Bonus_Interval(uint256) (CoxContract_flat.sol#1092-1095) is not in mixedCase
Parameter COX.setStake_Bonus_Interval(uint256).Stake_Bonus_Interval (CoxContract_flat.sol#1092) is not in mixedCase
Function COX.ViewStake_Bonus_Interval() (CoxContract_flat.sol#1097-1099) is not in mixedCase
Variable COX._enableStake_Bonus (CoxContract_flat.sol#311) is not in mixedCase
Variable COX._Burnt_Limit (CoxContract_flat.sol#313) is not in mixedCase
Variable COX._Min_Stake (CoxContract_flat.sol#314) is not in mixedCase
Variable COX._StartStakeBonusDate (CoxContract_flat.sol#315) is not in mixedCase
Variable COX._Entry_Days (CoxContract_flat.sol#316) is not in mixedCase
Variable COX._Stake_Bonus_Interval (CoxContract_flat.sol#317) is not in mixedCase
Variable COX._Scale (CoxContract_flat.sol#319) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
ReentrancyGuard._______gap (CoxContract_flat.sol#101) is never used in COX (CoxContract_flat.sol#284-1102)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
cycleUnstakeStatusOf(address,uint256) should be declared external:
        - COX.cycleUnstakeStatusOf(address,uint256) (CoxContract_flat.sol#721-723)
CheckStakeDetails() should be declared external:p]
[l;n' /.l?;
        - COX.CheckStakeDetails() (CoxContract_flat.sol#882-888)
viewTranferDetails() should be declared external:
        - COX.viewTranferDetails() (CoxContract_flat.sol#902-904)
minStake() should be declared external:
        - COX.minStake() (CoxContract_flat.sol#971-973)
stakeStats() should be declared external:
        - COX.stakeStats() (CoxContract_flat.sol#979-981)
viewEnableStakeBonus() should be declared external:
        - COX.viewEnableStakeBonus() (CoxContract_flat.sol#1060-1062)
ViewBurnLimit() should be declared external:
        - COX.ViewBurnLimit() (CoxContract_flat.sol#1069-1071)
ViewStartStakeBonusDate() should be declared external:
        - COX.ViewStartStakeBonusDate() (CoxContract_flat.sol#1079-1081)
ViewEntry_Days() should be declared external:
        - COX.ViewEntry_Days() (CoxContract_flat.sol#1088-1090)
ViewStake_Bonus_Interval() should be declared external:
        - COX.ViewStake_Bonus_Interval() (CoxContract_flat.sol#1097-1099)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:./CoxContract_flat.sol analyzed (4 contracts with 72 detectors), 81 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither raised one high severity issue related to state variable overshadowing which has already been covered in the Manual report. The medium issues raised are related to divide before multiply issues which are also covered in manual audit.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the COX contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Summary

Use case of the smart contract is simple and the code is relatively small. Altogether, the code is written and demonstrates effective use of abstraction, separation of concerns, and modularity. All high, medium and low severity issues have been fixed by the COX team and therefore the contract is good to be deployed on public networks as per the audit team's analysis.

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com