Crowfinance

Security Code Review

https://twitter.com/VidarTheAuditor - 1 February 2021

Overview

Project Summary

Project Name	Crowfinance
Description	Staking ecosystem
Platform	**Binance Smart Chain**, Solidity
Contracts	https://github.com/crowfinance/crow-smartcontract • **https://bscscan.com/address/0xB5ff2e278A1093850c4C0892b3b0FBF757BDbe67#code** • **https://bscscan.com/address/0x74800b7d796f34f92734514f9eb5b3d1b938cc1f#code** • **https://bscscan.com/address/0xcc2e12a9b5b75360c6fbf23b584c275d52cddb0e#code**

Executive Summary

Binance Smart Chain contracts were provided.

We have run extensive static analysis of the codebase as well as standard security assessment utilising industry approved tools.

There is no high level issues with the currently deployed contracts.

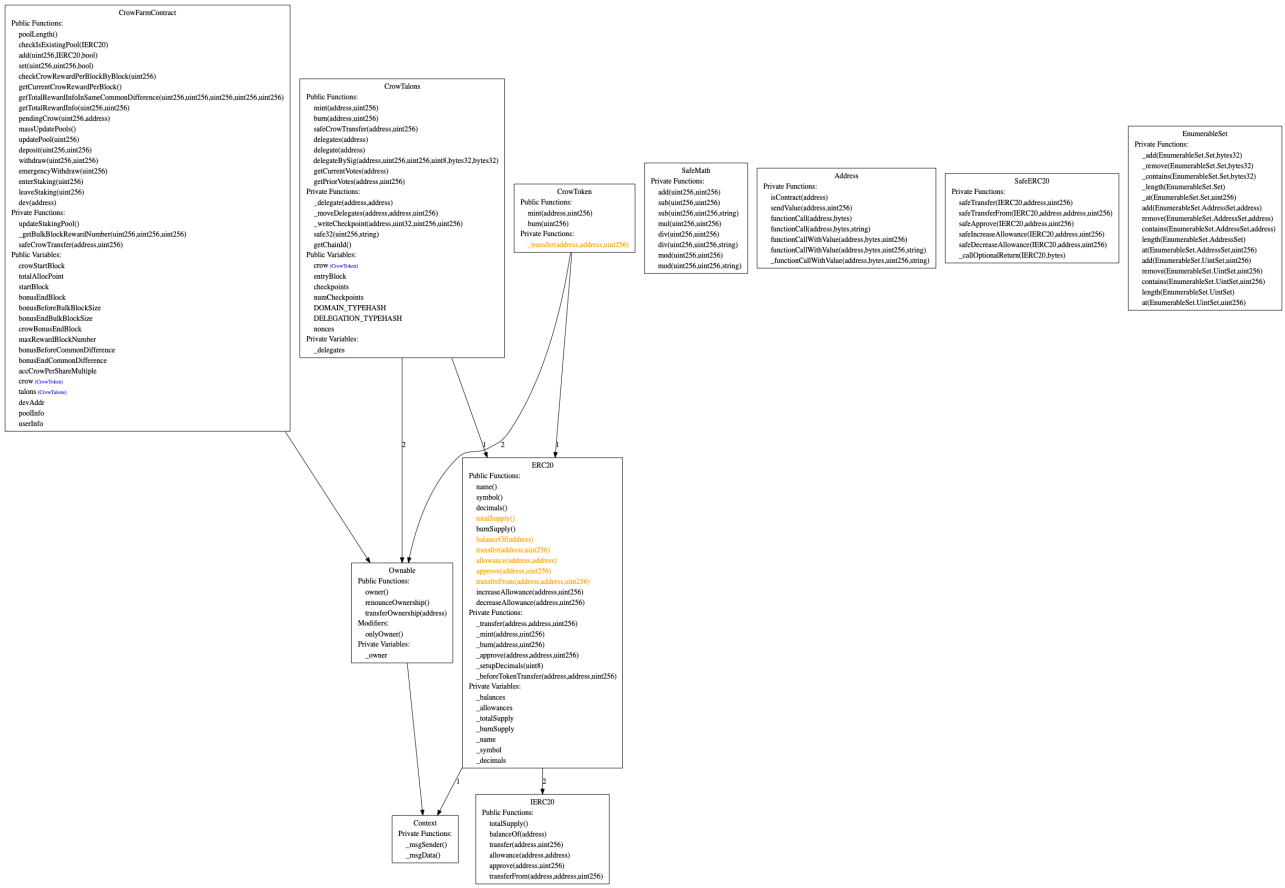
Some recommendations were issued, they are available in the report.

Disclaimer: The analysis did not include any tokenomics analysis (e.g. APY rates etc).

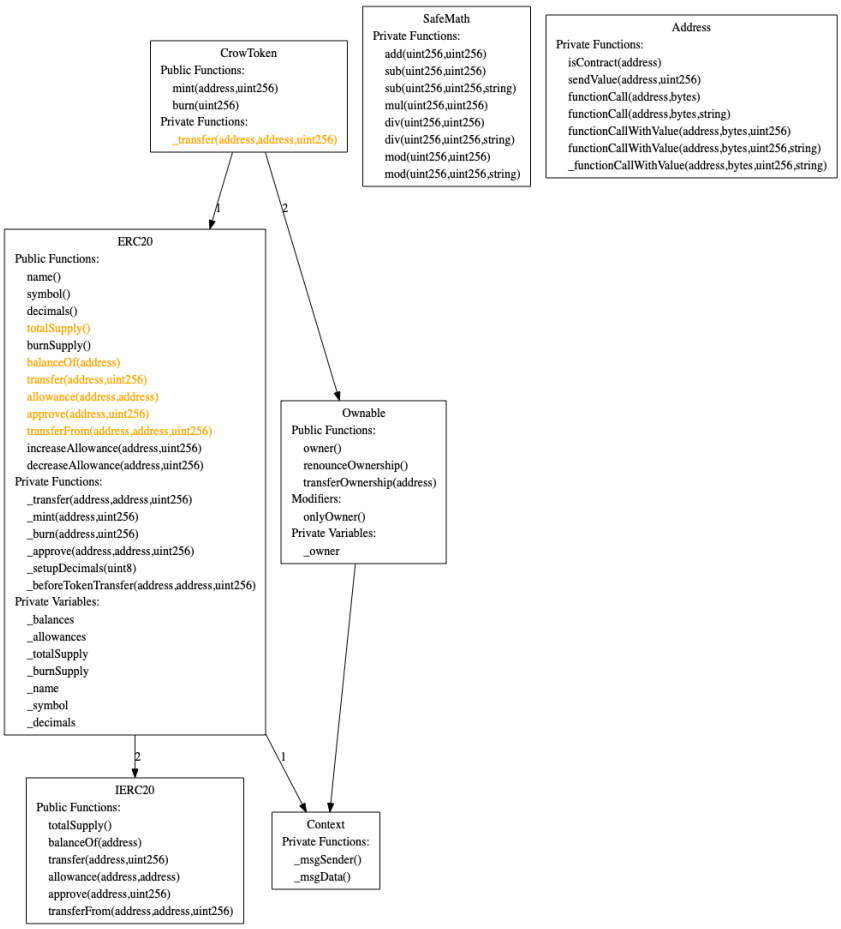
Architecture & Standards

Please find below the calling architecture of the reviewed contracts.

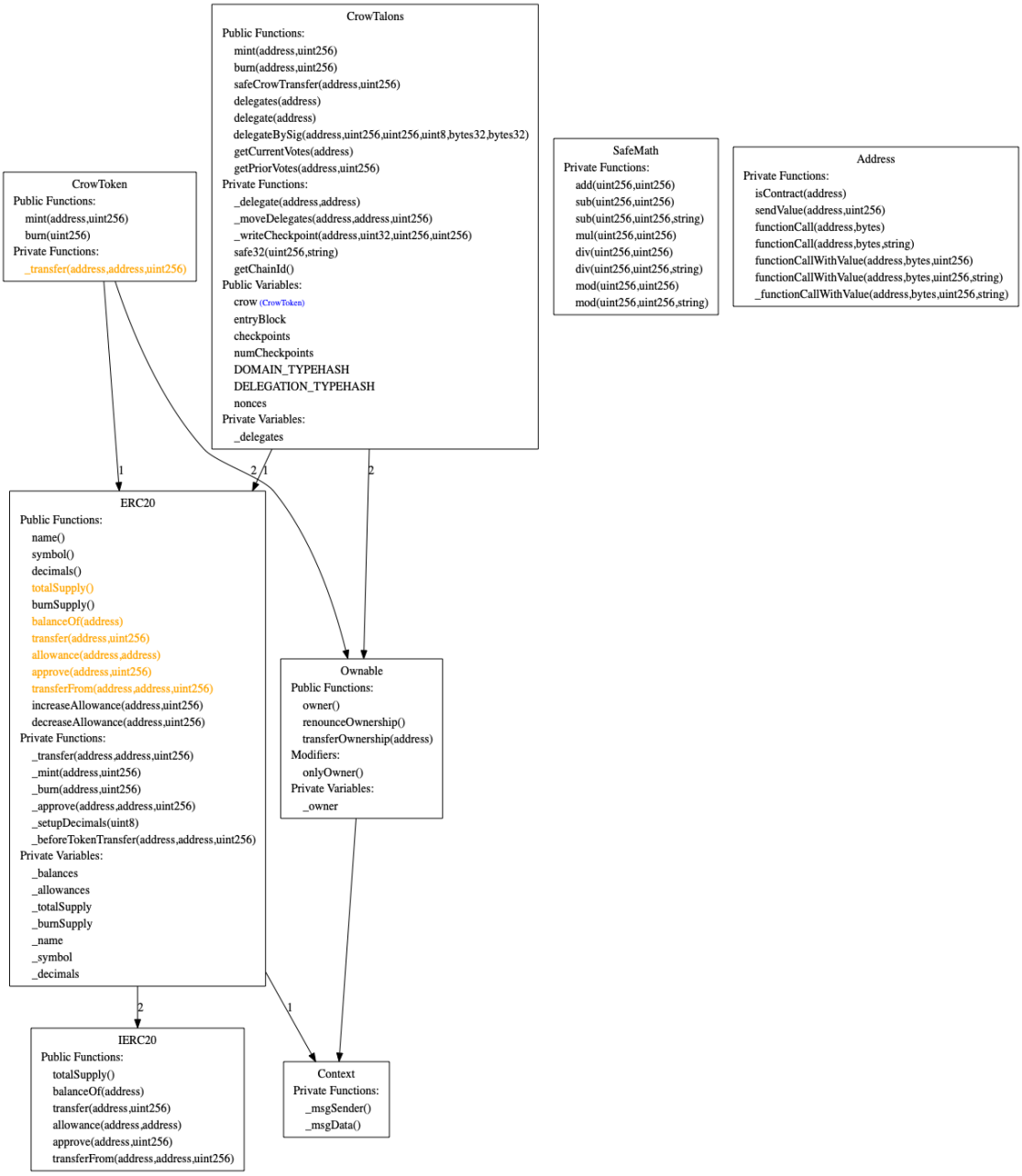
CrowFarmContract:



CrowToken:


SafeMath
Private Functions:
add(uint256,uint256)
sub(uint256,uint256)
sub(uint256,uint256,string)
mul(uint256,uint256)
div(uint256,uint256)
div(uint256,uint256,string)
mod(uint256,uint256)
mod(uint256,uint256,string)
Address
Private Functions:
isContract(address)
sendValue(address,uint256)
functionCall(address,bytes)
functionCall(address,bytes,string)
functionCallWithValue(address,bytes,uint256)
functionCallWithValue(address,bytes,uint256,string)
_functionCallWithValue(address,bytes,uint256,string)
CrowToken
Public Functions:
mint(address,uint256)
burn(uint256)
Private Functions:
_transfer(address,address,uint256)
1
2
ERC20
Public Functions:
name()
symbol()
decimals()
totalSupply()
burnSupply()
balanceOf(address)
transfer(address,uint256)
allowance(address,address)
approve(address,uint256)
transferFrom(address,address,uint256)
increaseAllowance(address,uint256)
decreaseAllowance(address,uint256)
Private Functions:
_transfer(address,address,uint256)
_mint(address,uint256)
_burn(address,uint256)
_approve(address,address,uint256)
_setupDecimals(uint8)
_beforeTokenTransfer(address,address,uint256)
Private Variables:
_balances
_allowances
_totalSupply
_burnSupply
_name
_symbol
_decimals
Ownable
Public Functions:
owner()
renounceOwnership()
transferOwnership(address)
Modifiers:
onlyOwner()
Private Variables:
_owner
2
1
IERC20
Public Functions:
totalSupply()
balanceOf(address)
transfer(address,uint256)
allowance(address,address)
approve(address,uint256)
transferFrom(address,address,uint256)
Context
Private Functions:
_msgSender()
_msgData()

CrowTalons:


CrowTalons
Public Functions:
mint(address,uint256)
burn(address,uint256)
safeCrowTransfer(address,uint256)
delegates(address)
delegate(address)
delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32)
getCurrentVotes(address)
getPriorVotes(address,uint256)
Private Functions:
_delegate(address,address)
_moveDelegates(address,address,uint256)
_writeCheckpoint(address,uint32,uint256,uint256)
safe32(uint256,string)
getChainId()
Public Variables:
crow (CrowToken)
entryBlock
checkpoints
numCheckpoints
DOMAIN_TYPEHASH
DELEGATION_TYPEHASH
nonces
Private Variables:
_delegates
SafeMath
Private Functions:
add(uint256,uint256)
sub(uint256,uint256)
sub(uint256,uint256,string)
mul(uint256,uint256)
div(uint256,uint256)
div(uint256,uint256,string)
mod(uint256,uint256)
mod(uint256,uint256,string)
Address
Private Functions:
isContract(address)
sendValue(address,uint256)
functionCall(address,bytes)
functionCall(address,bytes,string)
functionCallWithValue(address,bytes,uint256)
functionCallWithValue(address,bytes,uint256,string)
_functionCallWithValue(address,bytes,uint256,string)
CrowToken
Public Functions:
mint(address,uint256)
burn(uint256)
Private Functions:
_transfer(address,address,uint256)
1
2
1
2
ERC20
Public Functions:
name()
symbol()
decimals()
totalSupply()
burnSupply()
balanceOf(address)
transfer(address,uint256)
allowance(address,address)
approve(address,uint256)
transferFrom(address,address,uint256)
increaseAllowance(address,uint256)
decreaseAllowance(address,uint256)
Private Functions:
_transfer(address,address,uint256)
_mint(address,uint256)
_burn(address,uint256)
_approve(address,address,uint256)
_setupDecimals(uint8)
_beforeTokenTransfer(address,address,uint256)
Private Variables:
_balances
_allowances
_totalSupply
_burnSupply
_name
_symbol
_decimals
Ownable
Public Functions:
owner()
renounceOwnership()
transferOwnership(address)
Modifiers:
onlyOwner()
Private Variables:
_owner
2
1
IERC20
Public Functions:
totalSupply()
balanceOf(address)
transfer(address,uint256)
allowance(address,address)
approve(address,uint256)
transferFrom(address,address,uint256)
Context
Private Functions:
_msgSender()
_msgData()

CrowToken & CrowTalons contracts are fully BIP20 compatible.

```
# Check CrowToken

## Check functions
[✓] totalSupply() is present
        [✓] totalSupply() -> () (correct return value)
        [✓] totalSupply() is view
[✓] balanceOf(address) is present
        [✓] balanceOf(address) -> () (correct return value)
        [✓] balanceOf(address) is view
[✓] transfer(address,uint256) is present
        [✓] transfer(address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
        [✓] transferFrom(address,address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
        [✓] approve(address,uint256) -> () (correct return value)
        [✓] Approval(address,address,uint256) is emitted
[✓] allowance(address,address) is present
        [✓] allowance(address,address) -> () (correct return value)
        [✓] allowance(address,address) is view
[✓] name() is present
        [✓] name() -> () (correct return value)
        [✓] name() is view
[✓] symbol() is present
        [✓] symbol() -> () (correct return value)
        [✓] symbol() is view
[✓] decimals() is present
        [✓] decimals() -> () (correct return value)
        [✓] decimals() is view

## Check events
[✓] Transfer(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
[✓] Approval(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed


        [✓] CrowToken has increaseAllowance(address,uint256)
```

```
# Check CrowTalons

## Check functions
[✓] totalSupply() is present
        [✓] totalSupply() -> () (correct return value)
        [✓] totalSupply() is view
[✓] balanceOf(address) is present
        [✓] balanceOf(address) -> () (correct return value)
        [✓] balanceOf(address) is view
[✓] transfer(address,uint256) is present
        [✓] transfer(address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
        [✓] transferFrom(address,address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
        [✓] approve(address,uint256) -> () (correct return value)
        [✓] Approval(address,address,uint256) is emitted
[✓] allowance(address,address) is present
        [✓] allowance(address,address) -> () (correct return value)
        [✓] allowance(address,address) is view
[✓] name() is present
        [✓] name() -> () (correct return value)
        [✓] name() is view
[✓] symbol() is present
        [✓] symbol() -> () (correct return value)
        [✓] symbol() is view
[✓] decimals() is present
        [✓] decimals() -> () (correct return value)
        [✓] decimals() is view

## Check events
[✓] Transfer(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
[✓] Approval(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed


        [✓] CrowTalons has increaseAllowance(address,uint256)
```

Findings

Number of contracts: 11 (including inherited ones)

Use: SafeMath

Name	# functions	ERCS	ERC20 info	Complex code	Features
SafeMath	8			No	
Address	7			No	Send ETH Assembly
SafeERC20	6			No	Send ETH Tokens interaction
EnumerableSet	15			No	
CrowToken	34	ERC20	∞ Minting Approve Race Cond.	No	
CrowTalons	46	ERC20	∞ Minting Approve Race Cond.	Yes	Ecrecover Tokens interaction Assembly
CrowFarmContract	28			Yes	Tokens interaction

The minting on CrowToken and CrowTalons is part of the functionality of the system. As the owner of both contracts is CrowFarmContract it **does not** posses any risks.

For more info see Deployment section.

Static Analysis Findings

High issues: None

Medium issues:

Divide before multiply:

```
CrowFarmContract.getTotalRewardInfoInSameCommonDifference(uint256,uint256,uint256,uint256,uint256) (CrowFarmContract.sol#1639-1723) perform
 a multiplication on the result of a division:
        -N = tempCurrentPreviousBulkLastBlock.sub(lastRewardBulkLastBlock).div(tempBulkBlockSize) (CrowFarmContract.sol#1696)
        -totalReward = totalReward.add(N.mul(a1.add(aN)).div(2)) (CrowFarmContract.sol#1710)
CrowFarmContract.pendingCrow(uint256,address) (CrowFarmContract.sol#1764-1775) performs a multiplication on the result of a division:
        -crowReward = totalReward.mul(pool.allocPoint).div(totalAllocPoint) (CrowFarmContract.sol#1771)
        -accCrowPerShare = accCrowPerShare.add(crowReward.mul(accCrowPerShareMultiple).div(lpSupply)) (CrowFarmContract.sol#1772)
CrowFarmContract.updatePool(uint256) (CrowFarmContract.sol#1786-1805) performs a multiplication on the result of a division:
        -crowReward = totalReward.mul(pool.allocPoint).div(totalAllocPoint) (CrowFarmContract.sol#1800)
        -pool.accCrowPerShare = pool.accCrowPerShare.add(crowReward.mul(accCrowPerShareMultiple).div(lpSupply)) (CrowFarmContract.sol#1803)
```

In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

[Manual Check] It does not possesses significant risks for the contract.

Detection of the reentrancy:

```
Reentrancy in CrowFarmContract.leaveStaking(uint256) (CrowFarmContract.sol#1886-1903):
        External calls:
        - updatePool(0) (CrowFarmContract.sol#1890)
                - crow.mint(devAddr,crowReward.div(12)) (CrowFarmContract.sol#1801)
                - crow.mint(address(talons),crowReward) (CrowFarmContract.sol#1802)
        - safeCrowTransfer(msg.sender,pending) (CrowFarmContract.sol#1893)
                - talons.safeCrowTransfer(_to,_amount) (CrowFarmContract.sol#1907)
        State variables written after the call(s):
        - user.amount = user.amount.sub(_amount) (CrowFarmContract.sol#1896)
```

```
Reentrancy in CrowFarmContract.withdraw(uint256,uint256) (CrowFarmContract.sol#1828-1844):
        External calls:
        - updatePool(_pid) (CrowFarmContract.sol#1833)
                - crow.mint(devAddr,crowReward.div(12)) (CrowFarmContract.sol#1801)
                - crow.mint(address(talons),crowReward) (CrowFarmContract.sol#1802)
        - safeCrowTransfer(msg.sender,pending) (CrowFarmContract.sol#1836)
                - talons.safeCrowTransfer(_to,_amount) (CrowFarmContract.sol#1907)
        State variables written after the call(s):
        - user.amount = user.amount.sub(_amount) (CrowFarmContract.sol#1839)
```

[Manual Check] It is recommended to add nonReentrant modifier (https://docs.openzeppelin.com/contracts/2.x/api/utils#ReentrancyGuard-nonReentrant--)

More details see Manual Check section.

Low/Informational issues:

State variable could be declared constant:

```
CrowFarmContract.accCrowPerShareMultiple (CrowFarmContract.sol#1475) should be constant
```

State variable could be declared constant.

Block timestamps used:

```
CrowToken._transfer(address,address,uint256) (CrowFarmContract.sol#1123-1136) uses timestamp for comparisons
        Dangerous comparisons:
        - getHour >= 6 && getHour < 18 (CrowFarmContract.sol#1126)
CrowTalons.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (CrowFarmContract.sol#1253-1294) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(now <= expiry,CROW::delegateBySig: signature expired) (CrowFarmContract.sol#1292)
CrowFarmContract.enterStaking(uint256) (CrowFarmContract.sol#1858-1883) uses timestamp for comparisons
        Dangerous comparisons:
        - getHour >= 6 && getHour < 18 (CrowFarmContract.sol#1872)
```

Dangerous usage of block.timestamp. block.timestamp can be manipulated by miners.

[Manual Check] As it is required by the system functionality it does not posses significant risk.

Dynamic Tests

We have run fuzzing/property-based testing of Solidity smarts contracts. It was using sophisticated grammar-based fuzzing campaigns based on a contract ABI to falsify user-defined predicates or Solidity assertions.

There were also dynamic tests run on EVM byte code to detect common vulnerabilities including integer underflows, owner-overwrite-to-Ether-withdrawal, and others.

```
The analysis was completed successfully. No issues were detected.
```

No Issues were found.

Manual Checks

The following functions have **potential** re-entrancy vulnerabilities:

- CrowFarmContract.deposit(uint256,uint256)
- CrowFarmContract.enterStaking(uint256)
- CrowFarmContract.emergencyWithdraw(uint256)
- CrowFarmContract.leaveStaking(uint256)
- CrowFarmContract.withdraw(uint256,uint256)

[Recommendation] Use nonReentrant modifier (https://docs.openzeppelin.com/contracts/2.x/api/utils#ReentrancyGuard-nonReentrant--)

Please see Syntetix staking examples for the nonReentrant usage: https://github.com/Synthetixio/synthetix/blob/develop/contracts/StakingRewards.sol

Automatic Tests

The project lacks any automatic testing and tests scripts. We did not run any functional tests provided by the team, due to lack of such scripts provided. Hence the full business logic functionality was not tested.

[**Recommendation]**: Create comprehensive test cases and implement them as scripts or mocha tests using the hardhat infrastructure.

[**Disclaimer**] There were no tests conducted testing full system functionality due to lack of proper test cases and/or test scripts.

Deployment & Contract Ownership

The contracts are currently deployed on BSC Mainnet:

- https://bscscan.com/address/0xB5ff2e278A1093850c4C0892b3b0FBF757BDbe67#code
- https://bscscan.com/address/0x74800b7d796f34f92734514f9eb5b3d1b938cc1f#code
- https://bscscan.com/address/0xcc2e12a9b5b75360c6fbf23b584c275d52cddb0e#code

The owner of CrowToken and CrowTalons contracts is CrowFarmContract that provides the desired security based on minting functionality of the tokens.

The owner of CrowFarmContract is Timelock contract (https://bscscan.com/address/0x5E081D6B4BBf753c6cbf75d51c4e48e97c6C0740#code) that limits any changes to owner only parameters of CrowFarmContract limited to the following timelock delays:

- GRACE_PERIOD = 14 days;
- MINIMUM_DELAY = 12 hours;
- MAXIMUM_DELAY = 30 days;

Current delay of any change is set to 12 hours.

[Recommendation] As 12 hours is not long time considering community being based worldwide it is advisable to increase the delay to higher value.

Disclaimer

The information appearing in this report is for general purposes only and is not intended to provide any legal security guarantees to any individual or entity. As one review is not enough to provide 100% security against any attacks or bugs, it is advisable to conduct more reviews or/and audits.

The report does not provide personalised investment advice or recommendations, especially does not provide advice to conclude any transactions and it does not provide investment, financial, legal or tax advice.

We are not responsible or liable for any loss which results from the report.

The report should not be considered as an investment advice.