# CNXT TOKEN AUDIT

**February 2021**

## BLOCKCHAIN CONSILIUM

# Contents

# Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED "AS IS", WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND, AND BLOCKCHAIN CONSILIUM DISCLAIMS ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT. COPYRIGHT OF THIS REPORT REMAINS WITH BLOCKCHAIN CONSILIUM.

## Purpose of the report

The Audits and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the Solidity programming language that could present security risks. Cryptographic tokens and smart contracts are emergent technologies and carry with them high levels of technical risk and uncertainty.

The Audits are not an endorsement or indictment of any particular project or team, and the Audits do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Audits in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. This Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. There is no owed duty to any Third-Party by virtue of publishing these Audits.

# Introduction

We first thank Crowdnext for giving us the opportunity to audit their smart contract. This document outlines our methodology, audit details, and results.

Crowdnext asked us to review their CNXT token smart contract (ETH mainnet address: `0x64ac6dfd5d74c5a255c970f173feb37af6d6d04b`). Blockchain Consilium reviewed the system from a technical perspective looking for bugs, issues and vulnerabilities in their code base. The Audit is valid for `0x64ac6dfd5d74c5a255c970f173feb37af6d6d04b` Ethereum smart contract. The audit is not valid for any other versions of the smart contract. Read more below.

# Audit Summary

This code is clean, thoughtfully written and in general well architected. The code conforms closely to the documentation and specification. *We loved reading it.*

The code is based on OpenZeppelin in many cases. In general, OpenZeppelin's codebase is good, and this is a relatively safe start.

Overall, the code is well commented and clear on what it is supposed to do for each function. The visibility and state mutability of all the functions are clearly specified, and there are no confusions.

| **Audit Result** | ✓ **PASSED** |
|---|---|
| High Severity Issues | None |
| Moderate Severity Issues | None |
| Low Severity Issues | None |

# Overview

The project has one Solidity file for the CNXT ERC20 Token Smart Contract, the CrowdnextToken.sol file that contains about 504 lines of Solidity code. We manually reviewed each line of code in the smart contract. All the functions and state variables are well commented using the NatSpec documentation for the functions which is good to understand quickly how everything is supposed to work.

*Nice Features:*
The contract provides a good suite of functionality that will be useful for the entire

contract AND It **USES** SafeMath library to check for overflows and underflows, which protects against overflow and underflow attacks. All the ERC20 functions are included; it is a valid ERC20 token and in addition has some extra functionality for burning, minting, recovering tokens from contract, paying out dividends, locking and unlocking transfers and integrating with smart contracts / DAPPs.

### Methodology:

Blockchain Consilium manually reviewed the smart contract line-by-line, keeping in mind industry best practices and known attacks, looking for any potential issues and vulnerabilities, and areas where improvements are possible.

We also used automated tools like slither for analysis and reviewing the smart contract. The raw output of these tools is included in the Appendix. These tools often give false-positives, and any issues reported by them but not included in the issue list can be considered not valid.

### Classification / Issue Types Definition:

1. **High Severity:** which presents a significant security vulnerability or failure of the contract across a range of scenarios, or which may result in loss of funds.
2. **Moderate Severity:** which affects the desired outcome of the contract execution or introduces a weakness that can be exploited. It may not result in loss of funds but breaks the functionality or produces unexpected behaviour.
3. **Low Severity:** which does not have a material impact on the contract execution and is likely to be subjective.

The smart contract is considered to pass the audit, as of the audit date, if no high severity or moderate severity issues are found.

## Attacks & Issues considered while auditing

In order to check for the security of the contract, we reviewed each line of code in the smart contract considering several known Smart Contract Attacks & known issues.

- **Overflows and underflows:**
  An overflow happens when the limit of the type variable uint256 , 2 ** 256, is exceeded. What happens is that the value resets to zero instead of incrementing more.

For instance, if we want to assign a value to a uint bigger than 2 ** 256 it will simple go to 0—this is dangerous.

On the other hand, an underflow happens when you try to subtract 0 minus a number bigger than 0.For example, if you subtract 0 - 1 the result will be = 2 ** 256 instead of -1.

This is quite dangerous. This contract **DOES** check for overflows and underflows by using **OpenZeppelin's** *SafeMath*.

- **Reentrancy Attack:**
  One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. This class of bug can take many forms, and both of the major bugs that led to the DAO's collapse were bugs of this sort.

  This smart contract includes only two external calls in `approveAndCall` and `transferAnyERC20`, which are used to integrate with contracts, and recover any ERC20 tokens mistakenly sent to the token contract address, and is accessible only by the contract admin, no state changes take place after external calls, it follows checks-effects-interactions pattern, and thus *is not vulnerable* to re-entrancy attack.

- **Replay attack:**
  The replay attack consists of making a transaction on one blockchain like the original Ethereum's blockchain and then repeating it on another blockchain like the Ethereum's classic blockchain. The ether is transferred like a normal transaction from a blockchain to another. Though it's no longer a problem because since the version 1.5.3 of *Geth* and 1.4.4 of *Parity* both implement the attack protection EIP 155 by Vitalik Buterin.
  So the people that will use the contract depend on their own ability to be updated with those programs to keep themselves secure.

- **Short address attack:**
  This attack affects ERC20 tokens, was discovered by the Golem team and consists of the following:

  A user creates an Ethereum wallet with a trailing 0, which is not hard because it's only a digit. For instance: `0xiofa8d97756as7df5sd8f75g8675ds8gsdg0`
  Then he buys tokens by removing the last zero:
  Buy 1000 tokens from account `0xiofa8d97756as7df5sd8f75g8675ds8gsdg`. If the contract has enough amount of tokens and the buy function doesn't check the length of the address of the sender, the Ethereum's virtual machine will just add zeroes to the transaction until the address is complete.

The virtual machine will return 256000 for each 1000 tokens bought. This is abug of the virtual machine.

Here is a **fix for short address attacks**

```
modifier onlyPayloadSize(uint size) {
    assert(msg.data.length >= size + 4);
    _;
}
function transfer(address _to, uint256 _value) onlyPayloadSize(2 * 32) {
    // do stuff
}
```

*Whether or not it is appropriate for token contracts to mitigate the short-address attack is a contentious issue among smart-contract developers. Many, including those behind the OpenZeppelin project, have explicitly chosen not to do so. Blockchain Consilium doesn't consider short address attack an issue of the smart contract at the token level.*

This contract **does not** implement an `onlyPayloadSize(uint numwords)` modifier for `transfer`, `transferFrom`, `approve`, `increaseAllowance`, and `decreaseAllowance` functions, it probably assumes that checks for short address attacks are handled at a higher layer (which generally are), and since the `onlyPayloadSize()` modifier started causing some bugs restricting the flexibility of the smart contracts, it's alright not to check for short address attacks at the Token Contract level to allow for some more flexibility for dAPP coding, but the checks for short address attacks must be done at some layer of coding (e.g. for buys and sells, the exchange can do it - almost all well-known exchanges check for short address attacks after the Golem Team discovered it), this contract *does not prevent short address attack*, so the *checks for short address attack must be done while buying or selling or coding a DAPP using CNXT where necessary*.

You can read more about the attack here: ERC20 Short Address Attacks.

- **Approval Double-spend:**
  ERC20 Standard allows users to approve other users to manage their tokens, or spend tokens from their account till a certain amount, by setting the user's allowance with the standard `approve` function, then the allowed user may use `transferFrom` to spend the allowed tokens.

  Hypothetically, given a situation where Alice approves Bob to spend 100 Tokens from her account, and if Alice needs to adjust the allowance to allow Bob to spend 20 more tokens, normally – she'd check Bob's allowance (100 currently) and start a new `approve` transaction allowing Bob to spend a total of 120 Tokens instead of 100 Tokens.

Now, if Bob is monitoring the Transaction pool, and as soon as he observes new transaction from Alice approving more amount, he may send a `transferFrom` transaction spending 100 Tokens from Alice's account with higher gas price and do all the required effort to get his spend transaction mined before Alice's new approve transaction.

Now Bob has already spent 100 Tokens, and given Alice's approve transaction is mined, Bob's allowance is set to 120 Tokens, this would allow Bob to spend a total of 100 + 120 = 220 Tokens from Alice's account instead of the allowed 120 Tokens. This exploit situation is known as Approval Double-Spend Attack.

A potential solution to minimize these instances would be to set the non-zero allowance to 0 before setting it to any other amount.

It's possible for `approve` to enforce this behaviour without interface changes in the ERC20 specification:

```
if ((_value != 0) && (approved[msg.sender][_spender] != 0)) return false;
```

However, this is just an attempt to modify user behaviour. If the user does attempt to change from one non-zero value to another, the double spend might still happen, since the attacker may set the value to zero by already spending all the previously allowed value before the user's new approval transaction.

If desired, a non-standard function can be added to minimize hassle for users. The issue can be fixed with minimal inconvenience by taking a change value rather than a replacement value:

```
function increaseAllowance (address _spender, uint256 _addedValue)
returns (bool success) {
  uint oldValue = approved[msg.sender][_spender];
  approved[msg.sender][_spender] = safeAdd(oldValue, _addedValue);
  return true;
}
```

Even if this function is added, it's important to keep the original for compatibility with the ERC20 specification.

Likely impact of this bug is low for most situations. This contract implements an `increaseAllowance` and a `decreaseAllowance` function, both of which takes the change in value instead of taking the new value, which is really *nice*.

For more, see this discussion on GitHub:
https://github.com/ethereum/EIPs/issues/20#issuecomment263524729

- **Accidental Token Loss**

    - Token Smart Contracts should prevent transferring tokens to the token smart contract address if there's no good reason to not prevent, or if there's no way to take out tokens held by the token smart contract. The CNXT smart contract *does* prevent transferring of CNXT to CNXT smart contract address. If someone accidentally sends CNXT to the CNXT smart contract, the transaction will fail and thus their CNXT will be saved from getting lost.

    - In traditional ERC20 tokens, one more issue is when other ERC20 Tokens are transferred to the token smart contract, there would be no way to take them out, and this can be solved by implementing the "Any Token Transfer" improvement suggestion. This smart contract implements a `transferAnyERC20` function which is accessible only by admin, and is useful to recover tokens accidentally sent to CNXT Token Smart Contract Address (0x64Ac6Dfd5D74C5A255c970F173Feb37aF6d6D04B).

# Issues Found

## High Severity Issues

No high severity issues were found in the smart contract.

## Moderate Severity Issues

No moderate severity issues were found in the smart contract.

## Low Severity Issues

No low severity issues were found in the smart contract.

# Line by line comments

- Line 5:
  The compiler version is specified as 0.5.11, this means the code can be compiled with solidity compilers with the version 0.5.11, which is the latest version at the time of Auditing.

- Lines 7 to 69:
  SafeMath library is included for safe arithmetic operations.

- Lines 72 to 142:
  The `Ownable` contract makes the contract creator the owner of the contract, so that in `CrowdnextToken` the contract creator becomes the owner and receives the initially minted tokens, the `Ownable` contract has following noteworthy functions:

  1. `transferOwnership` allows the current owner transfer the control of the contract to a new Ethereum address when needed.

  2. `renounceOwnership` allows the current owner to relinquish control of the contract, if executed, it will not be possible to call owner-only functions anymore, for example, in this smart contract, if the owner renounces the ownership, they can no longer call the `mint, payDividends` or the `transferAnyERC20,` or locking functions.

- Lines 144 to 164:
  The ERC20 Standard Interface is included.

- Lines 166 to 432:
  1. A standard implementation of ERC20 standard along with some extra functions for locking / unlocking and distributing dividends is included.

  2. On line 189 `allowAddress` allows the owner to approve an address so that it can send out tokens even when all the tokens are locked.

  3. On line 194 `lockAddress` allows the owner to lock an address so that it cannot transfer tokens even when all the tokens are unlocked.

  4. On line 199 `setLocked` allows the owner to lock or unlock the token transfers.

5. On line 203 `canTransfer` is a readable function which is helpful in knowing whether a particular address can transfer CNXT tokens or not.

6. On line 222 `divsOwing` function calculates the number of dividends which a particular address can claim.

7. On line 227 `updateAccount` function sends out the dividends of a particular address and sets the divsOwing result for that address to 0.

8. On line 235, the `payDividends` function allows the owner to pay out dividends in Ether to all token holders based on their token shares.

9. The `claimDividends` function allows the token holders to claim their ETH dividends.

10. On line 334, the `increaseAllowance` function, and on line 349, the `decreaseAllowance` function are implemented to provide a measure against approval double spend attack.

11. On line 379, an internal `_mint` function is implemented to allow the owner to mint tokens.

12. On line 395, an internal `_burn` function is implemented to allow the token holders to burn there tokens.

13. On line 428, the internal `_burnFrom` function allows to burn tokens from a particular account which has approved the spender to transfer tokens from their account.

14. The `burn`, `burnFrom`, `transfer`, `transferFrom` functions all check whether the spender can transfer tokens first, for example, if all tokens are locked and the spender isn't specially allowed to transfer tokens, then they cannot transfer tokens. These functions also result in the sender and receiver (pending) dividends being automatically withdrawn (to their dividends balance, which can be manually claimed by the concerned token holder).

- Lines 434 to 455:
  The `ERC20Burnable` contract is implemented which will allow token holders to burn their tokens, this contract uses the internal `_burn` and `_burnFrom` functions mentioned above.

- Lines 457 to 472:

The `ERC20Mintable` contract is implemented which will allow the token holders to burn their tokens, burning the tokens will increase the holders balance as well as total supply.

- Lines 467 to 469:
  The `tokenRecipient` interface is included, which will allow implementation of one step `transferToContract` function or `approveAndCall` function in the `CrowdnextToken` contract.

- Lines 478 to 504.

    1. The `CrowdnextToken` implements the above `ERC20Mintable`, and `ERC20Burnable` contracts, assigns the name "Crowdnext", the symbol "CNXT", a supply of 100 Million CNXT, and 18 decimals. The constructor sends all initially minted tokens to owner.

    2. On line 490 the `approveAndCall` function allows the token to be integrated with smart contracts in such a way that the purchase / token transfer can be completed with the smart contract in one transaction, instead of the traditional ERC20 way which needs two transactions to be executed for the contract to know that the tokens arrived and now it should execute the purchase.

    3. On line 501, the `transferAnyERC20` function allows the owner to transfer out any other ERC20 tokens which might be sent to this token contract address by mistake, and thus it prevents locking up of ERC20 tokens in the contract in case any tokens are sent to it by mistake.

# Appendix

## Smart Contract Summary

- Contract SafeMath
  - From SafeMath
    - add(uint256,uint256) (internal)
    - div(uint256,uint256) (internal)
    - mod(uint256,uint256) (internal)
    - mul(uint256,uint256) (internal)
    - sub(uint256,uint256) (internal)
- Contract Ownable
  - From Ownable
    - isOwner() (public)
    - owner() (public)
    - renounceOwnership() (public)
    - transferOwnership(address) (public)
    - _transferOwnership(address) (internal)
    - constructor() (internal)
- Contract IERC20
  - From IERC20
    - allowance(address,address) (external)
    - approve(address,uint256) (external)
    - balanceOf(address) (external)
    - totalSupply() (external)
    - transfer(address,uint256) (external)
    - transferFrom(address,address,uint256) (external)
- Contract ERC20
  - From Ownable
    - isOwner() (public)
    - owner() (public)
    - renounceOwnership() (public)
    - transferOwnership(address) (public)
    - _transferOwnership(address) (internal)
    - constructor() (internal)
  - From IERC20
    - allowance(address,address) (external)
    - approve(address,uint256) (external)
    - balanceOf(address) (external)
    - totalSupply() (external)
    - transfer(address,uint256) (external)

        - transferFrom(address,address,uint256) (external)
    - From ERC20
        - allowAddress(address,bool) (public)
        - allowance(address,address) (public)
        - approve(address,uint256) (public)
        - balanceOf(address) (public)
        - canTransfer(address) (public)
        - claimDividends() (public)
        - decreaseAllowance(address,uint256) (public)
        - divsOwing(address) (public)
        - increaseAllowance(address,uint256) (public)
        - lockAddress(address,bool) (public)
        - payDividends() (public)
        - setLocked(bool) (public)
        - totalSupply() (public)
        - transfer(address,uint256) (public)
        - transferFrom(address,address,uint256) (public)
        - _approve(address,address,uint256) (internal)
        - _burn(address,uint256) (internal)
        - _burnFrom(address,uint256) (internal)
        - _mint(address,uint256) (internal)
        - _transfer(address,address,uint256) (internal)
        - slitherConstructorVariables() (internal)
        - updateAccount(address) (internal)
- Contract ERC20Burnable
    - From Ownable
        - isOwner() (public)
        - owner() (public)
        - renounceOwnership() (public)
        - transferOwnership(address) (public)
        - _transferOwnership(address) (internal)
        - constructor() (internal)
    - From IERC20
        - allowance(address,address) (external)
        - approve(address,uint256) (external)
        - balanceOf(address) (external)
        - totalSupply() (external)
        - transfer(address,uint256) (external)
        - transferFrom(address,address,uint256) (external)
    - From ERC20
        - allowAddress(address,bool) (public)
        - allowance(address,address) (public)
        - approve(address,uint256) (public)

        - balanceOf(address) (public)
        - canTransfer(address) (public)
        - claimDividends() (public)
        - decreaseAllowance(address,uint256) (public)
        - divsOwing(address) (public)
        - increaseAllowance(address,uint256) (public)
        - lockAddress(address,bool) (public)
        - payDividends() (public)
        - setLocked(bool) (public)
        - totalSupply() (public)
        - transfer(address,uint256) (public)
        - transferFrom(address,address,uint256) (public)
        - _approve(address,address,uint256) (internal)
        - _burn(address,uint256) (internal)
        - _burnFrom(address,uint256) (internal)
        - _mint(address,uint256) (internal)
        - _transfer(address,address,uint256) (internal)
        - updateAccount(address) (internal)
    - From ERC20Burnable
        - burn(uint256) (public)
        - burnFrom(address,uint256) (public)
        - slitherConstructorVariables() (internal)
- Contract ERC20Mintable
    - From Ownable
        - isOwner() (public)
        - owner() (public)
        - renounceOwnership() (public)
        - transferOwnership(address) (public)
        - _transferOwnership(address) (internal)
        - constructor() (internal)
    - From IERC20
        - allowance(address,address) (external)
        - approve(address,uint256) (external)
        - balanceOf(address) (external)
        - totalSupply() (external)
        - transfer(address,uint256) (external)
        - transferFrom(address,address,uint256) (external)
    - From ERC20
        - allowAddress(address,bool) (public)
        - allowance(address,address) (public)
        - approve(address,uint256) (public)
        - balanceOf(address) (public)
        - canTransfer(address) (public)

        - claimDividends() (public)
        - decreaseAllowance(address,uint256) (public)
        - divsOwing(address) (public)
        - increaseAllowance(address,uint256) (public)
        - lockAddress(address,bool) (public)
        - payDividends() (public)
        - setLocked(bool) (public)
        - totalSupply() (public)
        - transfer(address,uint256) (public)
        - transferFrom(address,address,uint256) (public)
        - _approve(address,address,uint256) (internal)
        - _burn(address,uint256) (internal)
        - _burnFrom(address,uint256) (internal)
        - _mint(address,uint256) (internal)
        - _transfer(address,address,uint256) (internal)
        - updateAccount(address) (internal)
    - From ERC20Mintable
        - mint(address,uint256) (public)
        - slitherConstructorVariables() (internal)
- Contract tokenRecipient
    - From tokenRecipient
        - receiveApproval(address,uint256,bytes) (external)
- Contract CrowdnextToken
    - From Ownable
        - isOwner() (public)
        - owner() (public)
        - renounceOwnership() (public)
        - transferOwnership(address) (public)
        - _transferOwnership(address) (internal)
        - constructor() (internal)
    - From IERC20
        - allowance(address,address) (external)
        - approve(address,uint256) (external)
        - balanceOf(address) (external)
        - totalSupply() (external)
        - transfer(address,uint256) (external)
        - transferFrom(address,address,uint256) (external)
    - From ERC20
        - allowAddress(address,bool) (public)
        - allowance(address,address) (public)
        - approve(address,uint256) (public)
        - balanceOf(address) (public)
        - canTransfer(address) (public)

        - claimDividends() (public)
        - decreaseAllowance(address,uint256) (public)
        - divsOwing(address) (public)
        - increaseAllowance(address,uint256) (public)
        - lockAddress(address,bool) (public)
        - payDividends() (public)
        - setLocked(bool) (public)
        - totalSupply() (public)
        - transfer(address,uint256) (public)
        - transferFrom(address,address,uint256) (public)
        - _approve(address,address,uint256) (internal)
        - _burn(address,uint256) (internal)
        - _burnFrom(address,uint256) (internal)
        - _mint(address,uint256) (internal)
        - _transfer(address,address,uint256) (internal)
        - updateAccount(address) (internal)
    - From ERC20Burnable
        - burn(uint256) (public)
        - burnFrom(address,uint256) (public)
    - From ERC20Mintable
        - mint(address,uint256) (public)
    - From CrowdnextToken
        - approveAndCall(address,uint256,bytes) (external)
        - constructor() (public)
        - transferAnyERC20(address,address,uint256) (public)
        - slitherConstructorVariables() (internal)

## Slither Results

```
> slither CrowdnextToken.sol

INFO:Detectors:
CrowdnextToken.transferAnyERC20(address,address,uint256) (CrowdnextToken.sol#501-
503) ignores return value by external calls
"IERC20(_tokenAddress).transfer(_to,_amount)" (CrowdnextToken.sol#502)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-
return
INFO:Detectors:
ERC20.allowAddress._allowed (local variable @ CrowdnextToken.sol#189) shadows:
        - ERC20._allowed (state variable @ CrowdnextToken.sol#256)
ERC20.balanceOf.owner (local variable @ CrowdnextToken.sol#272) shadows:
        - Ownable.owner (function @ CrowdnextToken.sol#94-96)
ERC20.allowance.owner (local variable @ CrowdnextToken.sol#282) shadows:
        - Ownable.owner (function @ CrowdnextToken.sol#94-96)
ERC20._approve.owner (local variable @ CrowdnextToken.sol#412) shadows:
```

```
        - Ownable.owner (function @ CrowdnextToken.sol#94-96)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-
variable-shadowing
INFO:Detectors:
Pragma version "0.5.11" allows old versions (CrowdnextToken.sol#5)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in ERC20.claimDividends() (CrowdnextToken.sol#243-252):
        -(success) = beneficiary.call.value(amountToSend)()
CrowdnextToken.sol#249
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-
level-calls
INFO:Detectors:
Parameter '_addr' of _addr (CrowdnextToken.sol#189) is not in mixedCase
Parameter '_allowed' of _allowed (CrowdnextToken.sol#189) is not in mixedCase
Parameter '_addr' of _addr (CrowdnextToken.sol#194) is not in mixedCase
Parameter '_locked' of _locked (CrowdnextToken.sol#194) is not in mixedCase
Parameter '_locked' of _locked (CrowdnextToken.sol#199) is not in mixedCase
Parameter '_addr' of _addr (CrowdnextToken.sol#203) is not in mixedCase
Parameter '_addr' of _addr (CrowdnextToken.sol#222) is not in mixedCase
Contract 'tokenRecipient' (CrowdnextToken.sol#474-476) is not in CapWords
Parameter '_spender' of _spender (CrowdnextToken.sol#490) is not in mixedCase
Parameter '_value' of _value (CrowdnextToken.sol#490) is not in mixedCase
Parameter '_extraData' of _extraData (CrowdnextToken.sol#490) is not in mixedCase
Parameter '_tokenAddress' of _tokenAddress (CrowdnextToken.sol#501) is not in
mixedCase
Parameter '_to' of _to (CrowdnextToken.sol#501) is not in mixedCase
Parameter '_amount' of _amount (CrowdnextToken.sol#501) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
CrowdnextToken.slitherConstructorVariables (CrowdnextToken.sol#478-504) uses
literals with too many digits:
        - initialSupply = 100000000 * (10 ** decimals)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-
digits
INFO:Detectors:
CrowdnextToken.initialSupply should be constant (CrowdnextToken.sol#484)
ERC20.pointMultiplier should be constant (CrowdnextToken.sol#214)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-
variables-that-could-be-declared-constant
INFO:Detectors:
Ownable.renounceOwnership() (CrowdnextToken.sol#120-123) should be declared
external
Ownable.transferOwnership(address) (CrowdnextToken.sol#129-131) should be declared
external
ERC20.allowAddress(address,bool) (CrowdnextToken.sol#189-192) should be declared
external
ERC20.lockAddress(address,bool) (CrowdnextToken.sol#194-197) should be declared
external
ERC20.setLocked(bool) (CrowdnextToken.sol#199-201) should be declared external
ERC20.payDividends() (CrowdnextToken.sol#235-241) should be declared external
ERC20.claimDividends() (CrowdnextToken.sol#243-252) should be declared external
ERC20.balanceOf(address) (CrowdnextToken.sol#272-274) should be declared external
IERC20.balanceOf(address) (CrowdnextToken.sol#157) should be declared external
IERC20.allowance(address,address) (CrowdnextToken.sol#159) should be declared
external
```

```
ERC20.allowance(address,address) (CrowdnextToken.sol#282-284) should be declared
external
ERC20.transfer(address,uint256) (CrowdnextToken.sol#291-294) should be declared
external
IERC20.transfer(address,uint256) (CrowdnextToken.sol#149) should be declared
external
IERC20.transferFrom(address,address,uint256) (CrowdnextToken.sol#153) should be
declared external
ERC20.transferFrom(address,address,uint256) (CrowdnextToken.sol#318-322) should be
declared external
ERC20.increaseAllowance(address,uint256) (CrowdnextToken.sol#334-337) should be
declared external
ERC20.decreaseAllowance(address,uint256) (CrowdnextToken.sol#349-352) should be
declared external
ERC20Burnable.burn(uint256) (CrowdnextToken.sol#443-445) should be declared
external
ERC20Burnable.burnFrom(address,uint256) (CrowdnextToken.sol#452-454) should be
declared external
ERC20Mintable.mint(address,uint256) (CrowdnextToken.sol#468-471) should be
declared external
CrowdnextToken.transferAnyERC20(address,address,uint256) (CrowdnextToken.sol#501-
503) should be declared external
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-
function-that-could-be-declared-as-external
INFO:Slither:CrowdnextToken.sol analyzed (8 contracts), 45 result(s) found
```