SOLIDITY.*FINANCE*

AXXA.ai Token - Audit Report

SUMMARY

AXXA intends to build a smart service and neural network ecosystem that runs powerful trading on cryptocurrency markets. For this audit we reviewed the project's token contract, deployed at 0x67d72156f9ee9b1a40da04021c20d54325e840ce. Further features are still in development.

Update - March 23rd, 2021 - The token has now been deployed on the Binance Smart Chain at 0xffca7c35339949f7b0af7267d00adff5f96e827e.

.

- *As of the time of the writing of this report, the circulating supply is 0 tokens. The owner of the contract has the ability to 'release' (mint) tokens up to the total supply cap of 1 billion. Only the owner has the ability to burn tokens.*
- *Ownership - Some functions are protected and can only be called by the contract owner. The owner can transfer ownership to any address.*
- *Ownership Protected functions release (mint) tokens up to the total supply, mark releasing finsihed (thereby preventing future minting), transfer ownership, and burn tokens.*
- *The owner additonally has the ability to recover any tokens erroneously sent to the contract address.*
- *Utilization of SafeMath to prevent overflows.*

Audit Findings Summary

- *No serious issues from external attackers were identified.*
- *Be aware of the ability of the owner to release tokens up to the total supply cap..*
- *Date: December 2nd, 2020.*
- *Update Date: March 23rd, 2021 - Add deployment to BSC.*

AUDIT RESULTS

We ran over 400,000 transactions interacting with this

Vulnerability Category	Notes	Result
Arbitrary Storage Write	N/A	PASS
Arbitrary Jump	N/A	PASS
Delegate Call to Untrusted Contract	N/A	PASS
Dependence on Predictable Variables	N/A	PASS
Deprecated Opcodes	N/A	PASS
Ether Thief	N/A	PASS
Exceptions	N/A	PASS
External Calls	N/A	PASS
Integer Over/Underflow	N/A	PASS
Multiple Sends	N/A	PASS
Suicide	N/A	PASS
State Change External Calls	N/A	Pass
Unchecked Retval	N/A	PASS

Vulnerability Category	Notes	Result
Critical Solidity Compiler	N/A	PASS
Overall Contract Safety		PASS

FUNCTION GRAPH

TRC20 Token Graph

INHERITENCE CHART

Multi-file Token

FUNCTIONS OVERVIEW

```
 ($) = payable function
 # = non-constant function

 Int = Internal
 Ext = External
 Pub = Public

 + [Lib] SafeMath
    - [Int] mul
    - [Int] div
    - [Int] sub
    - [Int] add

 +  ERC20Basic
```

```
+  ERC20 (ERC20Basic)
   - [Pub] allowance
   - [Pub] transferFrom #
   - [Pub] approve #

+  UnknownToken
   - [Pub] balanceOf
   - [Pub] transfer #

+ [Int] Token
   - [Ext] release #
   - [Ext] totalSupply
   - [Ext] balanceOf

+  AXXA (ERC20)
   - [Pub]  #
   - [Pub] transferOwnership #
      - modifiers: onlyOwner
   - [Pub] finishTokenRelease #
      - modifiers: onlyOwner,canRelease
   - [Prv] release #
      - modifiers: canRelease
   - [Pub] distributeAmounts #
      - modifiers: onlyOwner,canRelease
   - [Ext]  ($)
   - [Pub] balanceOf
   - [Pub] transfer #
      - modifiers: onlyPayloadSize
   - [Pub] transferFrom #
      - modifiers: onlyPayloadSize
   - [Pub] approve #
```

```
    - [Pub] recoverUnknownTokens #
       - modifiers: onlyOwner


 ($) = payable function
 # = non-constant function
```

SOURCE CODE

Click here to download the source code as a .sol file.

```
/**
 *Submitted for verification at Etherscan.io on 2020
*/

/*

AXXA.AI

*/
pragma solidity 0.4.20;

library SafeMath {
  function mul(uint256 a, uint256 b) internal pure r
    uint256 c = a * b;
    assert(a == 0 || c / a == b);
    return c;
  }
```

```
    return c;
  }

  function sub(uint256 a, uint256 b) internal pure r
    assert(b <= a);
    return a - b;
  }

  function add(uint256 a, uint256 b) internal pure r
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}

contract ERC20Basic {
    uint256 public totalSupply;
    function balanceOf(address who) public constant 
    function transfer(address to, uint256 value) pub
    event Transfer(address indexed from, address ind
}

contract ERC20 is ERC20Basic {
    function allowance(address owner, address spende
    function transferFrom(address from, address to, 
    function approve(address spender, uint256 value)
    event Approval(address indexed owner, address in
}

contract UnknownToken {
    function balanceOf(address _owner) constant publ
```

```
interface Token {
    function release(address _to, uint256 _value) ex
    function totalSupply() constant external returns
    function balanceOf(address _owner) constant exte
}

contract AXXA is ERC20 {

    using SafeMath for uint256;
    address owner = msg.sender;

    mapping (address => uint256) balances;
    mapping (address => mapping (address => uint256)

    string public constant name = "AXXA.AI";
    string public constant symbol = "AXXA";
    uint public constant decimals = 18;

    uint256 public totalSupply = 1000000000e18;
    uint256 public circulatingSupply = 0;
    uint256 public unreleasedTokens = totalSupply.su
    uint256 value;

    event Transfer(address indexed _from, address in
    event Approval(address indexed _owner, address i

    event Release(address indexed to, uint256 amount
    event ReleaseComplete();

    event Burn(address indexed burner, uint256 value
        require(!tokenReleaseComplete);
        _;
    }

    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }




    function AXXA () public {
        owner = msg.sender;
        release(owner, circulatingSupply);
    }

    function transferOwnership(address newOwner) onl
        owner = newOwner;
    }

    function finishTokenRelease() onlyOwner canRelea
        tokenReleaseComplete = true;
        ReleaseComplete();
        return true;
    }

    function release(address _to, uint256 _amount) c
        circulatingSupply = circulatingSupply.add(_a
        unreleasedTokens = unreleasedTokens.sub(_amo
        balances[_to] = balances[_to].add(_amount);
        Release(_to, _amount);
        if (circulatingSupply >= totalSupply) {
            tokenReleaseComplete = true;
        }
    }



    function distributeAmounts(address[] addresses,

        require(addresses.length <= 255);
        require(addresses.length == amounts.length);

        for (uint8 i = 0; i < addresses.length; i++)
            amounts[i]=amounts[i].mul(1e18); // no n
            require(amounts[i] <= unreleasedTokens);

            release(addresses[i], amounts[i]);

            if (circulatingSupply >= totalSupply) {
                tokenReleaseComplete = true;
            }
        }
    }

    function () external payable {

           owner.transfer(msg.value);
     }



    function balanceOf(address _owner) constant publ
            return balances[_owner];
    // mitigates the ERC20 short address attack
    modifier onlyPayloadSize(uint size) {
        assert(msg.data.length >= size + 4);
        _;
    }

    function transfer(address _to, uint256 _amount)

        require(_to != address(0));
        require(_amount <= balances[msg.sender]);

        balances[msg.sender] = balances[msg.sender].
        balances[_to] = balances[_to].add(_amount);
        Transfer(msg.sender, _to, _amount);
        return true;
    }

    function transferFrom(address _from, address _to

        require(_to != address(0));
        require(_amount <= balances[_from]);
        require(_amount <= allowed[_from][msg.sender

        balances[_from] = balances[_from].sub(_amoun
        allowed[_from][msg.sender] = allowed[_from][
        balances[_to] = balances[_to].add(_amount);
        Transfer(_from, _to, _amount);
        return true;
    }

    function approve(address _spender, uint256 _valu
        Approval(msg.sender, _spender, _value);
        return true;
    }

    function allowance(address _owner, address _spen
        return allowed[_owner][_spender];
    }

    function burn(uint256 _value) onlyOwner public {

        _value=_value.mul(1e18); // no need of decim
        require(_value <= balances[msg.sender]);
        // no need to require value <= totalSupply,
        // sender's balance is greater than the tota

        address burner = msg.sender;

        balances[burner] = balances[burner].sub(_val
        totalSupply = totalSupply.sub(_value);
        circulatingSupply = circulatingSupply.sub(_v
        Burn(burner, _value);
                Transfer(burner, address(0), _value)
    }

    function recoverUnknownTokens(address _tokenCont
        UnknownToken token = UnknownToken(_tokenCont
        uint256 amount = token.balanceOf(address(thi
        return token.transfer(owner, amount);
    }
```

GO HOME