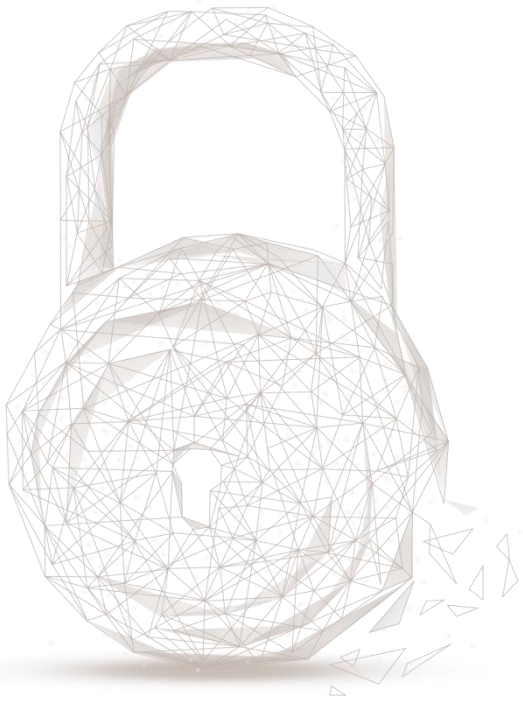
链安科技
Blockchain Security

Smart contract security audit report



Audit number : 201807082039

Smart Contract name :

CREDITS (CS)

Smart Contract address :

0x46b9Ad944d1059450Da1163511069C718F699D31

Smart Contract Link address :

https://etherscan.io/address/0x46b9ad944d1059450da1163511069c718f699d31#code

Start date of audit contract : 2018.07.05

Completion date of audit contract : 2018.07.08

Audit Conclusion : Pass (Good)

Audit team : Chengdu LianAn Technology Co. Ltd.

Audit type and results:

No.	Audit Type	Audit Subitems	Audit Results
1	Code For Programming Standardization Audit	ERC20 Token Standardization Audit	**Fail**
		Visibility Standardization Audit	Pass
		Gas Consumption Audit	Pass
		SafeMath Features Audit	Pass
		fallback Usage Audit	Pass
2	Function Call Audit	Function Call Permission Audit	Pass
		Call Function Security Audit	Pass
		Delegatecall Function Security Audit	Pass
		Self-destruct Function Security Audit	Pass
3	Integer Overflow/Underflow Audit	-	Pass
4	Reentrancy Attack Audit	-	Pass
5	Incorrect reachable state	-	Pass

6	Execution-Ordering Dependency Audit	-	Pass
7	Timestamp Dependency Audit	-	Pass
8	tx.origin Audit	-	Pass
9	Token Vault Audit	-	Pass

Note: Audit results and suggestions in code comments

Disclaimer : This audit is only for the range of audit types given in the audit result table. Other unknown security vulnerabilities are not within this auditing responsibility. Chengdu LianAn Technology issues this report only based on the vulnerabilities that have already occurred or existed and takes corresponding responsibility in this regard. As for the new attacks or vulnerabilities that occur or exist in the future, Chengdu LianAn Technology cannot determine the security status of its smart contracts and takes no responsibility for them. The security audit analysis and other contents of this report are only based on the documents and materials provided by the contract provider to Chengdu LianAn Technology as of the issued time of this report, and there are no missing, falsified, deleted, or concealed documents and materials. If the documents and materials provided are missing, falsified, deleted, concealed or inconsistent with the actual situation, Chengdu LianAn Technology does not take any responsibility to the losses and negative effects caused by this reason.

Explanations of audit results:

The audit result of the ERC20 Token Standardization Audit is not passed. The result is an irregular interface problem. The problem will not cause the contract to be maliciously exploited, but it may cause compatibility problem when the DAPP compiled by the higher version compiler or other smart contracts call the contract.

Detailed explanations of the failure of the audit results:

1.ERC20 Token Standardization Audit:

The transfer function defined in the CREDITS contract does not declare a return value. If the external contract compiled with Solidity 0.4.22 and later compilers calls the transfer() function according to the ABI analysis of the ERC20 standard, revert will occur.

```
131     /* Send coins */
132     function transfer(address _to, uint256 _value) public  {
133         require(((!Frozen&&AccountIsFrozen[msg.sender]!=true)||((Frozen)&&AccountIsNo
                AddressForReturn))&&now>AccountIsFrozenByDate[msg.sender]);
134         require(balanceOf[msg.sender] >= _value); // Check if the sender has enough
135         require (balanceOf[_to] + _value >= balanceOf[_to]); // Check for overflows
136         balanceOf[msg.sender] -= _value; // Subtract from the sender
137         balanceOf[_to] += _value; // Add the same to the recipient
138         Transfer(msg.sender, _to, _value); // Notify anyone listening that this trans
139         if (isHolder[_to] != true) {
140         Arrholders[Arrholders.length++] = _to;
141         isHolder[_to] = true;
142     }}
```

Figure 1 Transfer function declaration without returns

Contract source code audit notes:

```
pragma solidity ^ 0.4.19;   // Chengdu LianAn // It is recommended to use a fixed compiler version

contract Ownable {
    address public owner;
    function Ownable() public {
        owner = msg.sender;
    }

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



}

contract CREDITS is Ownable{
    /* Public variables of the token */
    string public name = 'CREDITS';
    string public symbol = 'CS';
    uint8 public decimals = 6;
    uint256 public totalSupply = 1000000000000000;
    uint public TotalHoldersAmount; // Chengdu LianAn // This state variable is not used in the contract, it is recommended to delete
    /*Freeze transfer from all accounts */
    bool public Frozen=true;
    bool public CanChange=true;
    address public Admin;
    address public AddressForReturn;
    address[] Accounts;  // Chengdu LianAn // This state variable is not used in the contract, it is recommended to delete
    /* This creates an array with all balances */
    mapping(address => uint256) public balanceOf;
```

```
    mapping(address => mapping(address => uint256)) public allowance;
   /*Individual Freeze*/
    mapping(address => bool) public AccountIsFrozen;
    /*Allow transfer for ICO, Admin accounts if IsFrozen==true*/
    mapping(address => bool) public AccountIsNotFrozen;
   /*Allow transfer tokens only to ReturnWallet*/
    mapping(address => bool) public AccountIsNotFrozenForReturn;
    mapping(address => uint) public AccountIsFrozenByDate; // Chengdu LianAn // The
unlocked date of the locked account address.

    mapping (address => bool) public isHolder;
    mapping (address => bool) public isArrAccountIsFrozen;
    mapping (address => bool) public isArrAccountIsNotFrozen;
    mapping (address => bool) public isArrAccountIsNotFrozenForReturn;
    mapping (address => bool) public isArrAccountIsFrozenByDate;// Chengdu LianAn //
Whether the account address exists in the locked account address array.
    address [] public Arrholders;
    address [] public ArrAccountIsFrozen;
    address [] public ArrAccountIsNotFrozen;
    address [] public ArrAccountIsNotFrozenForReturn;
    address [] public ArrAccountIsFrozenByDate;// Chengdu LianAn //  The locked account
address array.


    /* This generates a public event on the blockchain that will notify clients */
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
    event Burn(address indexed from, uint256 value);

//Chengdu LianAn // Judging the caller's frozen status
    modifier IsNotFrozen{

require(((!Frozen&&AccountIsFrozen[msg.sender]!=true)||((Frozen)&&AccountIsNotFrozen[msg.
sender]==true))&&now>AccountIsFrozenByDate[msg.sender]);
      _;
    }
```

```
    modifier isCanChange{
     require((msg.sender==owner||msg.sender==Admin)&&CanChange==true);
     _;
    }



    /* Initializes contract with initial supply tokens to the creator of the contract */

  function CREDITS() public {
        balanceOf[msg.sender] = totalSupply;
        Arrholders[Arrholders.length++]=msg.sender;
        Admin=msg.sender;
    }

     function setAdmin(address _address) public onlyOwner{
        require(CanChange);
        Admin=_address;
    }

   function setFrozen(bool _Frozen)public onlyOwner{
      require(CanChange);
      Frozen=_Frozen;
    }

    function setCanChange(bool _canChange)public onlyOwner{
      require(CanChange);  // Chengdu LianAn // Once CanChange is set to false, CanChange cannot be changed to true again, and owner or Admin can no longer execute setFrozen, setAccountIsFrozen, etc..
       CanChange=_canChange;
    }
    function setAccountIsFrozen(address _address, bool _IsFrozen)public isCanChange{
     AccountIsFrozen[_address]=_IsFrozen;
     if (isArrAccountIsFrozen[_address] != true) {
        ArrAccountIsFrozen[ArrAccountIsFrozen.length++] = _address;
        isArrAccountIsFrozen[_address] = true;
```

```
    }
    }
   function setAccountIsNotFrozen(address _address, bool _IsFrozen)public isCanChange{
        AccountIsNotFrozen[_address]=_IsFrozen;
        if (isArrAccountIsNotFrozen[_address] != true) {
            ArrAccountIsNotFrozen[ArrAccountIsNotFrozen.length++] = _address;
            isArrAccountIsNotFrozen[_address] = true;
        }
    }
   function setAccountIsNotFrozenForReturn(address _address, bool _IsFrozen)public
isCanChange{
        AccountIsNotFrozenForReturn[_address]=_IsFrozen;
        if (isArrAccountIsNotFrozenForReturn[_address] != true) {
            ArrAccountIsNotFrozenForReturn[ArrAccountIsNotFrozenForReturn.length++] =
_address;
            isArrAccountIsNotFrozenForReturn[_address] = true;
        }
    }
    function setAccountIsFrozenByDate(address _address, uint _Date)public isCanChange{

        require (!isArrAccountIsFrozenByDate[_address]);// Chengdu LianAn // Judging whether
the address has been added to the locked account address array.

        AccountIsFrozenByDate[_address]=_Date;// Chengdu LianAn // Setting account address
unlocked time.
        ArrAccountIsFrozenByDate[ArrAccountIsFrozenByDate.length++] = _address;// Chengdu
LianAn // Joining the locked account address array.
        isArrAccountIsFrozenByDate[_address] = true;// Chengdu LianAn // Changing account
address status to true

    }
/* Send coins */
    // Chengdu LianAn // Does not comply with the ERC20 interface standard.
    function transfer(address _to, uint256 _value) public  {// Chengdu LianAn // No returns value
declaration
        //Chengdu LianAn // It is recommended to check that the target address is not 0 to
avoid the loss of the token caused by operation error of the user.
```

```
        //Chengdu LianAn // Judging the sender's frozen status.
require(((!Frozen&&AccountIsFrozen[msg.sender]!=true)||((Frozen)&&AccountIsNotFrozen[msg.
sender]==true)||(AccountIsNotFrozenForReturn[msg.sender]==true&&_to==AddressForReturn))
&&now>AccountIsFrozenByDate[msg.sender]);
        require(balanceOf[msg.sender] >= _value); // Check if the sender has enough
        require (balanceOf[_to] + _value >= balanceOf[_to]); // Check for overflows
        balanceOf[msg.sender] -= _value; // Subtract from the sender
        balanceOf[_to] += _value; // Add the same to the recipient
        Transfer(msg.sender, _to, _value); // Notify anyone listening that this transfer took place
        if (isHolder[_to] != true) {
        Arrholders[Arrholders.length++] = _to;
        isHolder[_to] = true;
    }// Chengdu LianAn // If the receiver did not hold the token previously, add it to the
token owner array and set its token status to true.
}



    /* Allow another contract to spend some tokens in your behalf */
    // Chengdu LianAn // Beware that changing an allowance with this method brings the
risk that someone may use both the old and the new allowance by unfortunate transaction
ordering. One possible solution to mitigate this  race condition is to first reduce the
spender's allowance to 0 and set the desired value afterwards.
    function approve(address _spender, uint256 _value)public
    returns(bool success) {
        allowance[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
        return true;
    }
    /* A contract attempts to get the coins */
    function transferFrom(address _from, address _to, uint256 _value)public IsNotFrozen
returns(bool success)  {
        //Chengdu LianAn // It is recommended to check that the target address is not 0 to
avoid the loss of the token caused by operation error of the user.
        //Chengdu LianAn // Judging the sender's frozen status.

require(((!Frozen&&AccountIsFrozen[_from]!=true)||((Frozen)&&AccountIsNotFrozen[_from]==tr
```

```
ue))&&now>AccountIsFrozenByDate[_from]);

        require (balanceOf[_from] >= _value) ; // Check if the sender has enough
        require (balanceOf[_to] + _value >= balanceOf[_to]) ; // Check for overflows
        require (_value <= allowance[_from][msg.sender]) ; // Check allowance
        balanceOf[_from] -= _value; // Subtract from the sender
        balanceOf[_to] += _value; // Add the same to the recipient
        allowance[_from][msg.sender] -= _value;
        Transfer(_from, _to, _value);
        if (isHolder[_to] != true) {
        Arrholders[Arrholders.length++] = _to;
        isHolder[_to] = true;
        }//Chengdu LianAn // If the receiver did not hold the token previously, add it to the
token owner array and set its token status to true.

        return true;
    }
 /* @param _value the amount of money to burn*/

    function burn(uint256 _value) public IsNotFrozen  returns (bool success) {
        require(balanceOf[msg.sender] >= _value);   // Check if the sender has enough
        balanceOf[msg.sender] -= _value;            // Subtract from the sender
        totalSupply -= _value;                      // Updates totalSupply
        Burn(msg.sender, _value);
        return true;
    }
     /* Destroy tokens from other account  */

    function burnFrom(address _from, uint256 _value) public IsNotFrozen returns (bool success) {

require(((!Frozen&&AccountIsFrozen[_from]!=true)||((Frozen)&&AccountIsNotFrozen[_from]==tr
ue))&&now>AccountIsFrozenByDate[_from]);
        require(balanceOf[_from] >= _value);                // Check if the targeted balance is enough
        require(_value <= allowance[_from][msg.sender]);    // Check allowance
        balanceOf[_from] -= _value;                         // Subtract from the targeted balance
        allowance[_from][msg.sender] -= _value;             // Subtract from the sender's allowance
        totalSupply -= _value;                              // Update totalSupply
```

```
    Burn(_from, _value);
    return true;
}



function GetHoldersCount () public view returns (uint _HoldersCount){

    return (Arrholders.length-1);
}

function GetAccountIsFrozenCount () public view returns (uint _Count){

    return (ArrAccountIsFrozen.length);
}

function GetAccountIsNotFrozenForReturnCount () public view returns (uint _Count){

    return (ArrAccountIsNotFrozenForReturn.length);
}

function GetAccountIsNotFrozenCount () public view returns (uint _Count){

    return (ArrAccountIsNotFrozen.length);
}

 function GetAccountIsFrozenByDateCount () public view returns (uint _Count){

    return (ArrAccountIsFrozenByDate.length);
}

 function SetAddressForReturn (address _address) public isCanChange  returns (bool success ){
    AddressForReturn=_address;
    return true;
}

function setSymbol(string _symbol) public onlyOwner {
```

```
        require(CanChange);
        symbol = _symbol;
    }

    function setName(string _name) public onlyOwner {
        require(CanChange);
        name = _name;
    }


    /* This unnamed function is called whenever someone tries to send ether to it */
    function () public payable {
        revert();
    }
}
```

链安科技

Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/LianAnTech