CIBUS TOKEN Testing Details

About:

This document provides the details of testing of CIBUS Token smart contract file.

Token Information:

1. **Contract type:** ERC-20 standard Token
2. **Test environment:** Rinkeby testnet (Ethereum Test platform)
3. **Contract Address:** 0x3de5885d97828f9c11ef9476307e3b86d25b9e61
4. **Etherscan link:**

https://rinkeby.etherscan.io/token/0x3de5885d97828f9c11ef9476307e3b86d25b9e61

Contract Information:

1. Contract is tested with following credential:
   Token name: CIBUS Token Test4
   Token Symbol: CBTTEST4
   Decimal points accepted: 10

2. **Contract Source Code:**

```
pragma solidity ^0.4.19;

/*
 * Abstract Token Smart Contract.  Copyright © 2017 by CIBUS WORLD.
 * Author: contact@cibus.world
 */

 /*
 * Safe Math Smart Contract.  Copyright © 2017-2018 by CIBUS WORLD.
 * Author: contact@cibus.world
 * https://github.com/OpenZeppelin/zeppelin-
solidity/blob/master/contracts/math/SafeMath.sol
 */

contract SafeMath {
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
            if (a == 0) {
            return 0;
            }
            uint256 c = a * b;
            assert(c / a == b);
```

```
        return c;
    }

    function safeDiv(uint256 a, uint256 b) internal pure returns (uint256) {
        // assert(b > 0); // Solidity automatically throws when dividing by 0
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold
        return c;
    }

    function safeSub(uint256 a, uint256 b) internal pure returns (uint256) {
        assert(b <= a);
        return a - b;
    }

    function safeAdd(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        assert(c >= a);
        return c;
    }
}
/**
 * ERC-20 standard token interface, as defined
 * <a href="http://github.com/ethereum/EIPs/issues/20">here</a>.
 */
contract Token {

    function totalSupply() constant returns (uint256 supply);
    function balanceOf(address _owner) constant returns (uint256 balance);
    function transfer(address _to, uint256 _value) returns (bool success);
    function transferFrom(address _from, address _to, uint256 _value) returns (bool success);
    function approve(address _spender, uint256 _value) returns (bool success);
    function allowance(address _owner, address _spender) constant returns (uint256 remaining);
    event Transfer(address indexed _from, address indexed _to, uint256 _value);
    event Approval(address indexed _owner, address indexed _spender, uint256 _value);
}

/**
 * Abstract Token Smart Contract that could be used as a base contract for
 * ERC-20 token contracts.
 */
contract AbstractToken is Token, SafeMath {
    /**
     * Create new Abstract Token contract.
     */
    function AbstractToken () {
        // Do nothing
    }

    /**
     * Get number of tokens currently belonging to given owner.
     *
     * @param _owner address to get number of tokens currently belonging to the
     *        owner of
     * @return number of tokens currently belonging to the owner of given address
```

```
 */
function balanceOf(address _owner) constant returns (uint256 balance) {
        return accounts [_owner];
}

/**
 * Transfer given number of tokens from message sender to given recipient.
 *
 * @param _to address to transfer tokens to the owner of
 * @param _value number of tokens to transfer to the owner of given address
 * @return true if tokens were transferred successfully, false otherwise
 * accounts [_to] + _value > accounts [_to] for overflow check
 * which is already in safeMath
 */
function transfer(address _to, uint256 _value) returns (bool success) {
        require(_to != address(0));
        if (accounts [msg.sender] < _value) return false;
        if (_value > 0 && msg.sender != _to) {
        accounts [msg.sender] = safeSub (accounts [msg.sender], _value);
        accounts [_to] = safeAdd (accounts [_to], _value);
        }
        Transfer (msg.sender, _to, _value);
        return true;
}

/**
 * Transfer given number of tokens from given owner to given recipient.
 *
 * @param _from address to transfer tokens from the owner of
 * @param _to address to transfer tokens to the owner of
 * @param _value number of tokens to transfer from given owner to given
 *        recipient
 * @return true if tokens were transferred successfully, false otherwise
 * accounts [_to] + _value > accounts [_to] for overflow check
 * which is already in safeMath
 */
function transferFrom(address _from, address _to, uint256 _value)
returns (bool success) {
        require(_to != address(0));
        if (allowances [_from][msg.sender] < _value) return false;
        if (accounts [_from] < _value) return false;

        if (_value > 0 && _from != _to) {
   allowances [_from][msg.sender] = safeSub (allowances [_from][msg.sender], _value);
        accounts [_from] = safeSub (accounts [_from], _value);
        accounts [_to] = safeAdd (accounts [_to], _value);
        }
        Transfer(_from, _to, _value);
        return true;
}

/**
 * Allow given spender to transfer given number of tokens from message sender.
 * @param _spender address to allow the owner of to transfer tokens from message sender
 * @param _value number of tokens to allow to transfer
```

```
     * @return true if token transfer was successfully approved, false otherwise
     */
    function approve (address _spender, uint256 _value) returns (bool success) {
            allowances [msg.sender][_spender] = _value;
            Approval (msg.sender, _spender, _value);
            return true;
    }

    /**
     * Tell how many tokens given spender is currently allowed to transfer from
     * given owner.
     *
     * @param _owner address to get number of tokens allowed to be transferred
     *        from the owner of
     * @param _spender address to get number of tokens allowed to be transferred
     *        by the owner of
     * @return number of tokens given spender is currently allowed to transfer
     *        from given owner
     */
    function allowance(address _owner, address _spender) constant
    returns (uint256 remaining) {
            return allowances [_owner][_spender];
    }

    /**
     * Mapping from addresses of token holders to the numbers of tokens belonging
     * to these token holders.
     */
    mapping (address => uint256) accounts;

    /**
     * Mapping from addresses of token holders to the mapping of addresses of
     * spenders to the allowances set by these token holders to these spenders.
     */
    mapping (address => mapping (address => uint256)) private allowances;
}


/**
 * CIBUS token smart contract.
 */
contract CIBUSToken is AbstractToken {
    /**
     * Maximum allowed number of tokens in circulation.
     *Total Supply for Pre ICO = (10% of 100000000 = 10000000)  and  ICO = (30% of 100000000 =
30000000)
     * 10^^10 is done for decimal places, this is standard practice as all ethers are actually wei in
EVM
     */


    uint256 constant MAX_TOKEN_COUNT = 100000000 * (10**10);

    /**
     * Address of the owner of this smart contract.
```

```
 */
address private owner;

/**
 * Current number of tokens in circulation.
 */
uint256 tokenCount = 0;


/**
 * True if tokens transfers are currently frozen, false otherwise.
 */
bool frozen = false;

/**
 * Counter of total funds collected, in wei
*/
uint public totalCollected = 0;

/**
 * Create new CIBUS token smart contract and make msg.sender the
 * owner of this smart contract.
 */
function CIBUSToken () {
        owner = msg.sender;
}

/**
 * Get total number of tokens in circulation.
 *
 * @return total number of tokens in circulation
 */
function totalSupply() constant returns (uint256 supply) {
        return tokenCount;
}

string constant public name = "CIBUS Token Test4";
string constant public symbol = "CBTTEST4";
uint8 constant public decimals = 10;

/**
 * Transfer given number of tokens from message sender to given recipient.
 *
 * @param _to address to transfer tokens to the owner of
 * @param _value number of tokens to transfer to the owner of given address
 * @return true if tokens were transferred successfully, false otherwise
 */
function transfer(address _to, uint256 _value) returns (bool success) {
        if (frozen) return false;
        else return AbstractToken.transfer (_to, _value);
}

/**
 * Transfer given number of tokens from given owner to given recipient.
 *
```

```
 * @param _from address to transfer tokens from the owner of
 * @param _to address to transfer tokens to the owner of
 * @param _value number of tokens to transfer from given owner to given
 *        recipient
 * @return true if tokens were transferred successfully, false otherwise
 */
function transferFrom(address _from, address _to, uint256 _value)
        returns (bool success) {
        if (frozen) return false;
        else return AbstractToken.transferFrom (_from, _to, _value);
}

 /**
 * Change how many tokens given spender is allowed to transfer from message
 * spender.  In order to prevent double spending of allowance,
 * To change the approve amount you first have to reduce the addresses`
 * allowance to zero by calling `approve(_spender, 0)` if it is not
 * already 0 to mitigate the race condition described here:
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
 * @param _spender address to allow the owner of to transfer tokens from
 *        message sender
 * @param _value number of tokens to allow to transfer
 * @return true if token transfer was successfully approved, false otherwise
 */
function approve (address _spender, uint256 _value)
        returns (bool success) {
  require(allowance (msg.sender, _spender) == 0 || _value == 0);
        return AbstractToken.approve (_spender, _value);
}

/**
 * Create _value new tokens and give new created tokens to msg.sender.
 * May only be called by smart contract owner.
 *
 * @param _value number of tokens to create
 * @param _collected total amounts of fund collected for this issuance, in wei
 * @return true if tokens were created successfully, false otherwise
 */
function createTokens(uint256 _value, uint _collected)
        returns (bool success) {
        require (msg.sender == owner);

        if (_value > 0) {
        if (_value > safeSub (MAX_TOKEN_COUNT, tokenCount)) return false;

        accounts [msg.sender] = safeAdd (accounts [msg.sender], _value);
        tokenCount = safeAdd (tokenCount, _value);
        totalCollected = safeAdd(totalCollected, _collected);

   // adding transfer event and _from address as null address
   Transfer(0x0, msg.sender, _value);

   return true;
        }
```

```
    return false;

}


/**
 * For future use only whne we will need more tokens for our main application
 * Create mintedAmount new tokens and give new created tokens to target.
 * May only be called by smart contract owner.
 * @param mintedAmount number of tokens to create
 * @return true if tokens were created successfully, false otherwise
 */

function mintToken(address target, uint256 mintedAmount)
returns (bool success) {
        require (msg.sender == owner);
        if (mintedAmount > 0) {

        accounts [target] = safeAdd (accounts [target], mintedAmount);
        tokenCount = safeAdd (tokenCount, mintedAmount);

    // adding transfer event and _from address as null address
    Transfer(0x0, target, mintedAmount);

        return true;
        }
    return false;

        }

/**
 * Set new owner for the smart contract.
 * May only be called by smart contract owner.
 *
 * @param _newOwner address of new owner of the smart contract
 */
function setOwner(address _newOwner) {
        require (msg.sender == owner);

        owner = _newOwner;
}

/**
 * Freeze token transfers.
 * May only be called by smart contract owner.
 */
function freezeTransfers () {
        require (msg.sender == owner);

        if (!frozen) {
        frozen = true;
        Freeze ();
        }
}
```

```
 /**
  * Unfreeze token transfers.
  * May only be called by smart contract owner.
  */
 function unfreezeTransfers () {
        require (msg.sender == owner);

        if (frozen) {
        frozen = false;
        Unfreeze ();
        }
 }

 /*A user is able to unintentionally send tokens to a contract
  * and if the contract is not prepared to refund them they will get stuck in the contract.
  * The same issue used to happen for Ether too but new Solidity versions added the payable
modifier to
  * prevent unintended Ether transfers. However, there's no such mechanism for token transfers.
  * so the below function is created
  */

 function refundTokens(address _token, address _refund, uint256 _value) {
        require (msg.sender == owner);
        require(_token != address(this));
        AbstractToken token = AbstractToken(_token);
        token.transfer(_refund, _value);
        RefundTokens(_token, _refund, _value);
 }

 /**
  * Logged when token transfers were frozen.
  */
 event Freeze ();

 /**
  * Logged when token transfers were unfrozen.
  */
 event Unfreeze ();

 /**
  * when accidentally send other tokens are refunded
  */

 event RefundTokens(address _token, address _refund, uint256 _value);
}
```

Contract ABI:


```
[{
    "constant": false,
    "inputs": [],
    "name": "freezeTransfers",
    "outputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
}, {
    "constant": true,
    "inputs": [],
    "name": "name",
    "outputs": [{
            "name": "",
            "type": "string"
    }],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
}, {
    "constant": false,
    "inputs": [{
            "name": "_spender",
            "type": "address"
    }, {
            "name": "_value",
            "type": "uint256"
    }],
    "name": "approve",
    "outputs": [{
            "name": "success",
            "type": "bool"
    }],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
}, {
    "constant": false,
    "inputs": [{
            "name": "_newOwner",
            "type": "address"
    }],
    "name": "setOwner",
    "outputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
}, {
    "constant": true,
    "inputs": [],
```

```
    "name": "totalSupply",
    "outputs": [{
            "name": "supply",
            "type": "uint256"
    }],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
}, {
    "constant": false,
    "inputs": [{
            "name": "_from",
            "type": "address"
    }, {
            "name": "_to",
            "type": "address"
    }, {
            "name": "_value",
            "type": "uint256"
    }],
    "name": "transferFrom",
    "outputs": [{
            "name": "success",
            "type": "bool"
    }],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
}, {
    "constant": true,
    "inputs": [],
    "name": "decimals",
    "outputs": [{
            "name": "",
            "type": "uint8"
    }],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
}, {
    "constant": false,
    "inputs": [],
    "name": "unfreezeTransfers",
    "outputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
}, {
    "constant": false,
    "inputs": [{
            "name": "_value",
            "type": "uint256"
    }, {
            "name": "_collected",
            "type": "uint256"
```

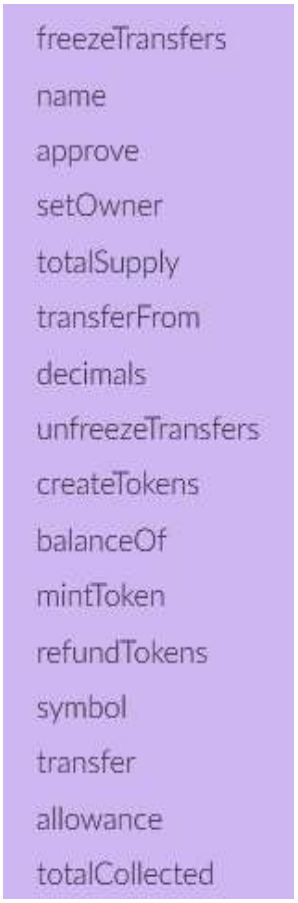
```
    }],
    "name": "createTokens",
    "outputs": [{
            "name": "success",
            "type": "bool"
    }],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
}, {
    "constant": true,
    "inputs": [{
            "name": "_owner",
            "type": "address"
    }],
    "name": "balanceOf",
    "outputs": [{
            "name": "balance",
            "type": "uint256"
    }],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
}, {
    "constant": false,
    "inputs": [{
            "name": "target",
            "type": "address"
    }, {
            "name": "mintedAmount",
            "type": "uint256"
    }],
    "name": "mintToken",
    "outputs": [{
            "name": "success",
            "type": "bool"
    }],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
}, {
    "constant": false,
    "inputs": [{
            "name": "_token",
            "type": "address"
    }, {
            "name": "_refund",
            "type": "address"
    }, {
            "name": "_value",
            "type": "uint256"
    }],
    "name": "refundTokens",
    "outputs": [],
    "payable": false,
```

```
    "stateMutability": "nonpayable",
    "type": "function"
}, {
    "constant": true,
    "inputs": [],
    "name": "symbol",
    "outputs": [{
            "name": "",
            "type": "string"
    }],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
}, {
    "constant": false,
    "inputs": [{
            "name": "_to",
            "type": "address"
    }, {
            "name": "_value",
            "type": "uint256"
    }],
    "name": "transfer",
    "outputs": [{
            "name": "success",
            "type": "bool"
    }],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
}, {
    "constant": true,
    "inputs": [{
            "name": "_owner",
            "type": "address"
    }, {
            "name": "_spender",
            "type": "address"
    }],
    "name": "allowance",
    "outputs": [{
            "name": "remaining",
            "type": "uint256"
    }],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
}, {
    "constant": true,
    "inputs": [],
    "name": "totalCollected",
    "outputs": [{
            "name": "",
            "type": "uint256"
    }],
```

```
    "payable": false,
    "stateMutability": "view",
    "type": "function"
}, {
    "inputs": [],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "constructor"
}, {
    "anonymous": false,
    "inputs": [],
    "name": "Freeze",
    "type": "event"
}, {
    "anonymous": false,
    "inputs": [],
    "name": "Unfreeze",
    "type": "event"
}, {
    "anonymous": false,
    "inputs": [{
            "indexed": false,
            "name": "_token",
            "type": "address"
    }, {
            "indexed": false,
            "name": "_refund",
            "type": "address"
    }, {
            "indexed": false,
            "name": "_value",
            "type": "uint256"
    }],
    "name": "RefundTokens",
    "type": "event"
}, {
    "anonymous": false,
    "inputs": [{
            "indexed": true,
            "name": "_from",
            "type": "address"
    }, {
            "indexed": true,
            "name": "_to",
            "type": "address"
    }, {
            "indexed": false,
            "name": "_value",
            "type": "uint256"
    }],
    "name": "Transfer",
    "type": "event"
}, {
    "anonymous": false,
    "inputs": [{
```

```
            "indexed": true,
            "name": "_owner",
            "type": "address"
        }, {
            "indexed": true,
            "name": "_spender",
            "type": "address"
        }, {
            "indexed": false,
            "name": "_value",
            "type": "uint256"
        }],
        "name": "Approval",
        "type": "event"
    }]
```

3. Public functions:



4. Actual ethereum mainnet contact will have different name and symbols as per CIBUS WORLD specification. All functionality will remain same.

5. **Document information:**
   The document provides testing of different methods with positive and negative scenarios.
   TX hash has been provided as a provenance, which resides in Rinke by test net Blockchain.
   To check transaction example:
https://rinkeby.etherscan.io/tx/0x9c2cebfec24f756954627252191639e45c0b39a0bedfdf8cfb7e301e59c748ce

Test cases:

Test case 1:

Scenario:

Owner access to generate tokens

1. **Create Token** from 0xeA4387E32e312C1D5141565f59c9F514f3C08ca2 (current Owner)
Tx Hash: 0x9c2cebfec24f756954627252191639e45c0b39a0bedfdf8cfb7e301e59c748ce
Result: Success
2. **Mint Token** from 0xeA4387E32e312C1D5141565f59c9F514f3C08ca2 (current Owner)
to 0x1Bb47c36F34dB4DC246E8AA870A31cf79f829555
Tx Hash: 0x18e2ec513e2d4064c29c8d354001db0b586b2500d73952f391ffe913613aa84e
Result: Success

Test case 2:

Scenario:

Other user wants to create token

3. **Create Token** from 0x8ff965527b29ABc895e76eE0E2504816f82F5b95 (not owner)
Tx Hash: 0xa837a77c58cf07afda8160ae15fc32d37941956a49a46811bd7a75eee604e86c
Result: Fail
Analysis: This is intended behaviour as other user cannot create or mint token directly.
4. **Mint Token** from 0x8ff965527b29ABc895e76eE0E2504816f82F5b95 (not owner)
Tx Hash: 0xb97c1d470234913de5ca71b716eae285dfc636f5266d2482b8af5f6e44cd0ef6
Result: Fail
Analysis: This is intended behaviour as other user cannot create or mint token directly.

Test case 3:

Scenario:

Change the owner

5. **setOwner** from 0x8ff965527b29ABc895e76eE0E2504816f82F5b95 (not owner)
Tx Hash: 0x00bc08a551c40a373719e3ce5296f179aa37095dfd2eb6163ecf2d18f647ba4f
Result: Fail
Analysis: This is intended behaviour as other user cannot change the owner, only current owner can change the ownership.
5. **setOwner** from 0xeA4387E32e312C1D5141565f59c9F514f3C08ca2 (owner)
Tx Hash:
0xd74377a7307bfc9f2c50a820816d35e5b500be5b8f393812e5923aae8a640660
Result: Success
Analysis: This is intended behaviour as only current owner can change the ownership.

Test case 4:

Scenario:

New owner can create or mintokens

6. **Create Token** from 0x8ff965527b29ABc895e76eE0E2504816f82F5b95 (current owner)
Tx Hash: 0xe9682afafff05abbf3e50bd8096cc2a71d8887b6af69d38e5d857ff8ff721245
Result: Success
Analysis: This is intended behavior, current owner can create or mint tokens.
Note: Specify high gas value for 20 Gwei gas price specify 71000 gas limit.

7. **Mint Token** from 0x8ff965527b29ABc895e76eE0E2504816f82F5b95 (current owner)
Tx Hash: 0x15ad73296baf519e40bc7cb31e2f1e66e2521fe0078285d15227ca4ab085af20
Result: Success
Analysis: This is intended behaviour, current owner can create or mint tokens.

Test Case 5:

Scenario:

Check token balance of an address

8. **Check balance** (CBT Token balance) of account
Read method and working good

No tx Hash is specified as it is read function so block is mined for it.

Test case 6:

Scenario:

Total Supply of tokens at this point of time.

9. **Total Supply of Tokens.**
It will return the number of total tokens created till now in the contract.
Read method and working good
No tx Hash is specified as it is read function so block is mined for it.

Test Case 7:

Scenario:

Token Transfer process

10. **Transfer** from 0xeA4387E32e312C1D5141565f59c9F514f3C08ca2 (current owner)
Tx Hash: 0xa1f6f27e46e4d55933dad1a8582c2e7a66b409accd58a3d16237f5bf007f8173
Result: Success
Analysis: This is intended behaviour any token holder can transfer tokens to other account when token freeze flag is false. By default it is false.
Note: gas limit will be 41000 and gas price 21Gwei.

11.**Transfer** in freeze situation from 0x8ff965527b29ABc895e76eE0E2504816f82F5b95 (not owner)
Tx Hash:
0x81d7c321ec0cb3b8c63fac3840eb84cc72ccd8c731bb7e48826b319cb280a5e4
Result: Fail
Analysis: This is intended behaviour any token holder cannot transfer tokens to other account when token freeze flag is true.
12. **Transfer** in unfreeze situation from
0x8ff965527b29ABc895e76eE0E2504816f82F5b95 (not owner)
Tx Hash: 0x7b0b3541c3f18bafc90618bc1c6c42e34f483adc318b868f382909d4b7c2866f
Result: Success
Analysis: This is intended behaviour, token holder can transfer tokens to other account when token freeze flag is false.
Note: Gas limit will be 41000 and gas price 21Gwei.

Test case 8:

Scenario:

freeze and unfreeze token transfer

13. **freeze transfer** from 0xeA4387E32e312C1D5141565f59c9F514f3C08ca2 (current owner)
Tx Hash: 0x37d792a1e1b5bcfdea7fc3c32b7a7ae5ae01a37ae78f80048e8d340968cabae7
Result: Success
Analysis: Only owner can freeze the token transfer. Token transfer is in freeze state now. By default it was false, now it is true.
Note: Gas limit must be 51000 and gas price: 21 Gewi

14. **unfreeze transfer** from 0x8ff965527b29ABc895e76eE0E2504816f82F5b95 (not owner)
Tx Hash: 0x83dd2c5e8aa32c226f121d0cb764b30a997c928cc9ccff07b7ccb60dd1d501e8
Result: Fail
Analysis: Only owner can freeze or unfreeze the token transfer.

15. **freeze transfer** from 0x8ff965527b29ABc895e76eE0E2504816f82F5b95 (not owner)
Tx Hash: 0x0cee995b536c53372feecf26513ab2b45d605af7bc026d1f75aa48b4d3cdf17a
Result: Fail
Analysis: Only owner can freeze or unfreeze the token transfer.

Test case 9:

Scenario:

Approve to send the token on behalf of another token Holder

16. **Approve** from 0xeA4387E32e312C1D5141565f59c9F514f3C08ca2 to 0x8ff965527b29ABc895e76eE0E2504816f82F5b95

Tx Hash: 0x3771a1c46199feec3650b449571e94cddc8977885d81a70ad2aa27814252773e
Result: Success
Analysis: Now delegated account can send the specified amount on behalf of approver token holder.
Note: Gas limit is 51000 and gas price 21Gwei.

17. **Check allowance** from 0xeA4387E32e312C1D5141565f59c9F514f3C08ca2 to 0x8ff965527b29ABc895e76eE0E2504816f82F5b95
Read method and status Success.

18. **Change approval** amount from 0xeA4387E32e312C1D5141565f59c9F514f3C08ca2 to 0x8ff965527b29ABc895e76eE0E2504816f82F5b95

Tx Hash: 0x281f88970281744e0b84d5e170542f4ab64a8ee8b09c88a620caa5e3d383fcaf
Result: Fail
Analysis: cannot change the approval amount unless make it 0 and allocate fresh value. This is to avoid race condition in the code.
Note: Gas limit is 51000 and gas price 21Gwei.

19. **Making 0 and then set approval** from 0xeA4387E32e312C1D5141565f59c9F514f3C08ca2 to 0x8ff965527b29ABc895e76eE0E2504816f82F5b95

Tx Hash: 0xba356932aba4a0dc6594b2cb6d3af5e73abc3c63ddb54d2ca6624b6277f0b25a (to make 0) and
0x82d1223b85e4603b908854c939eddda7620290b58b1c74c576cf904a4bb55af1 (to make new approval amount)
Result: Success
Analysis: Now delegated account can send the specified amount on behalf of approver token holder.
Note: Gas limit is 51000 and gas price 21Gwei.

Test case 10:

Scenario:

Transfer from 'part' or 'full' allowance amount to a different account.

20. **Transfer from** 0xeA4387E32e312C1D5141565f59c9F514f3C08ca2 to 0x1Bb47c36F34dB4DC246E8AA870A31cf79f829555
by 0x8ff965527b29ABc895e76eE0E2504816f82F5b95
Tx Hash: 0xdd07a85acf09773a4783a26584e4baf981f0e2aab83b86680727b5d8c592acc4
Result: Success
Analysis: it is intended behaviour as the amount can be examined allowance function.
Note: Gas limit is 51000 and gas price 21Gwei.

20. **Transfer excess than delegated amount from**
0xeA4387E32e312C1D5141565f59c9F514f3C08ca2
to
0x1Bb47c36F34dB4DC246E8AA870A31cf79f829555
by
0x8ff965527b29ABc895e76eE0E2504816f82F5b95

Tx Hash: 0xa95d7c5b6cca1dab1edf1080fbed21792e3ac5fec89825d68cd7b33ea6ac3803
Result: Fail
Analysis: More than delegated amount cannot be transferred
Note: Gas limit is 51000 and gas price 21Gwei.

Test case 11:

Scenario:

Accidentally send other tokens locked into the contract, so refund Tokens function is used to send back to the user.

21. **Refund Tokens** from 0xeA4387E32e312C1D5141565f59c9F514f3C08ca2 via the contract.

Tx Hash:

0x538c0ed99f1ce57d2960fe362b6ed4ac4e6a270429a7915f4cf646d0c456b043

Result: Success

Analysis: Different token called CIBUS TOKEN TEST2 was transferred to the contract. which was now in the contract address, now owner sends back the other type token to accidentally send token holder's account.

Note: Gas limit is 61000 and gas price 21Gwei.