JohnWick Sec

Guard the internet of value

ETH SMART CONTRACT AUDIT REPORT

JOHNWICK SECURITY LAB

WWW.JOHNWICK.IO

John Wick Security Lab received the **EBASE** (company/team) **EURBASE Stablecoin (EBASE)** project smart contract code audit requirements on **2019/08/29.**

Project Name：EURBASE Stablecoin (EBASE)

Smart Contract Address：

https://etherscan.io/address/0x86fadb80d8d2cff3c3680819e4da99c10232ba0f#code

Audit Number：201900807

Audit Date：20190830

Audit Category and Result:

Category	Sub-category	Result（Pass/Not Pass）
Contract vulnerability	Integer overflow	Pass
	Race condition	Not Pass
	Denial of service	Pass
	Logical vulnerability	Pass
	Hardcoded address	Pass
	Function input parameter check	Not Pass
	Function access control bypass	Pass
	Random number generation	Pass
	Random number use	Pass
Contract specification	Solidity compiler version	Pass
	Event use	Pass
	fallback function use	Pass
	Constructor use	Pass
	Function visibility declaration	Pass
	Variable storage declaration	Pass
	Deprecated keyword use	Pass
	ERC20/223 standard	Pass
	ERC721 standard	Pass
Business risk	Able to arbitrarily create token	Not Pass
	Able to arbitrarily destroy token	Pass
	Able to arbitrarily suspend tx.	Note
	"Short address" attack	Pass
	"Fake recharge" attack	Pass
GAS optimization	assert()/require()	Pass
	Loop(for/while) optimization	Pass
	Storage optimization	Pass
Automated fuzzing		Pass

(Other unknown security vulnerabilities and Ethereum design flaws are not included in this audit responsibility)

Audit Result: **PASS** (The contract owner can suspend any transfer transaction at any time)

Auditor: John Wick Security Lab

(**Disclaimer:** The John Wick Security Lab issues this report based on the facts that have occurred or existed before the issuance of this report and assumes corresponding responsibility in this regard. For the facts that occur or exist after the issuance of this report, the John Wick Security Lab cannot judge the security status of its smart contracts and does not assume any responsibility for it. The safety audit analysis and other contents of this report are based on the relevant materials and documents provided by the information provider to the John Wick Security Lab when the report is issued (referred to as the information provided). The John Wick Security Lab assumes that there is no missing, falsified, deleted, or concealed information provided. If the information provided is missing, falsified, deleted, concealed, or the information provider's response is inconsistent with the actual situation, the John Wick Security Lab shall not bear any responsibility for the resulting loss and adverse effects.)

Audit Details:

This smart contract is written based on the **ERC223** standard. It implements the standard interfaces and also implements functions such as ownable, blacklistable, pausable, mintable, burnable and so on.

Token basic information
- contract address: 0x86FADb80d8D2cff3C3680819E4da99C10232Ba0F
- creator address: 0x736C3D674121Eb0F6AE182cbA6fC30368a2FD933
- blacklister address: 0x736C3D674121Eb0F6AE182cbA6fC30368a2FD933
- owner address: 0x1C7B838045225Ef10DF9Da6D240C194F2171EAe1
- Token name: EURBASE Stablecoin
- Token symbol: EBASE
- Token decimals: 18
- Token initial supply: 1,000,000000000000000000
- Token total supply: 5,000,000000000000000000 (up to now)

//JohnWick: 122L 205L 441L

The contract class uses the `constructor` keyword (introduced in solidity 0.4.22) to implement the constructor, in line with the recommended practice.

//JohnWick: 263L
This contract uses the `SafeMath` library to avoid potential integer overflow issues and is in line with recommended practices.

//JohnWick: 392L
According to the ERC20 specification, after creating a new token, the contract should trigger the `Transfer` event that transfers the token to `address(0)`, which is in line with the recommended practice.

//JohnWick: 431L
After destroying the token, the function `_burn(address _who, uint256 _value)` triggers the `Transfer` event that transfers the token from `_who` to `address(0)`, which is in line with the recommended practice.

//JohnWick: 438L
The decimal point of this contract is 18, which is consistent with the decimal point of the Ethereum base currency ETH, which is in line with the recommended practice.

//JohnWick: **[Low Risk]** 283L
The function `approve(address _spender, uint256 _value)` has a race condition problem.
We recommend adding the following check code after 283 line:
`require(_value == 0 || allowed[msg.sender][_spender] == 0);`
Or use the `increaseApproval` or `decreaseApproval` functions of this contract to achieve atomic increase or decrease `allowed[msg.sender][_spender]` to avoid this problem.

Vulnerability Analysis: The `approve` function is generally used to authorize other accounts/contracts to how many tokens can be withdrawn. For example:
At the beginning, `msg.sender` has authorized `spender` can spend 100 tokens, Then `msg.sender` thinks 100 tokens are too much, and calls the `approve` function again that tries to reduce the allowance to 60 tokens.
However, `spender` observed this transaction before the transaction was packaged, so `spender` initiated the transaction of withdrawing 100 tokens, and by rising the **gas price**, the miner preferentially packaged spender's transaction and successfully got 100 tokens.
Then `msg.sender`'s allowance authorization transaction is done, and the

`spender` initiates the transaction of withdrawing 60 tokens, so that the `spender` finally got 160 tokens.

Therefore, we recommend adding the following check code to the approve function:
`require(_value == 0 || allowed[msg.sender][_spender] == 0);`

Every time `msg.sender` modify the allowance, he must first set the allowance to zero, and then change the allowance to the corresponding value in order to avoid the potential race condition problem.

`//JohnWick:` **`[Low Risk]`** `388L`
The function `mint(address _to, uint256 _amount) onlyOwner canMint` does not limit the token's total supply. And the value of `mintingFinished` is `false` up to now, which means that the function modifier `canMint` allows to mint tokens. If the contract `owner` issues the token unlimitedly, it will damage the interests of the previous investors.

We recommend adding the following check code after 388 line, in which the storage variable `maxTotalSupply` is the upper limit of the token's total supply :
`require(totalSupply.add(_amount) <= maxTotalSupply);`

`//JohnWick:` **`[Low Risk]`** `388L`
The function `mint(address _to, uint256 _amount) onlyOwner canMint` does not check if the `_to` account is address(0). If the contract `owner` issues the token incorrectly to address(0), the token will be lost.

`//JohnWick:` **`[Note]`**
`transfer,approve/increaseApproval/decreaseApproval/transferFrom` functions have used the function modifier `whenNotPaused`.
In other words, the contract `owner` can suspend all transfer transactions at any time, which needs related digital asset exchanges to pay attention to.
Solution: The contract `owner` can set the owner to a private key uncontrolled account such as address(1) by calling the `transferOwnership(address newOwner)` to give up the right to suspend/resume the transfer transaction.

Note: The line number of the code involved in the audit details is based

on the verified contract source code uploaded by the project party at etherscan.io, which is also displayed as a backup in the Smart Contract Source Code section of this report.

Smart Contract Source Code:

```
/**
 *Submitted for verification at Etherscan.io on 2019-08-20
*/

pragma solidity ^0.5.7;

interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address who) external view returns (uint256);
    function allowance(address owner, address spender) external view returns
(uint256);

    function transfer(address to, uint256 value) external returns (bool);
    function approve(address spender, uint256 value) external returns (bool);
    function transferFrom(address from, address to, uint256 value) external
returns (bool);

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256
value);
}

interface IERC223 {
    function name() external view returns (string memory);
    function symbol() external view returns (string memory);
    function decimals() external view returns (uint8);
    function totalSupply() external view returns (uint256);

    function balanceOf(address who) external view returns (uint);

    function transfer(address to, uint value) external returns (bool);
    function transfer(address to, uint value, bytes calldata data) external
returns (bool);

    event Transfer(address indexed from, address indexed to, uint value, bytes
indexed data);
}

contract ContractReceiver {
```

```
    function tokenFallback(address _from, uint _value, bytes memory _data)
public {

    }
}

/**
 * @title SafeMath
 * @dev Unsigned math operations with safety checks that revert on error.
 */
library SafeMath {
    /**
      * @dev Multiplies two unsigned integers, reverts on overflow.
      */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero,
but the
        // benefit is lost if 'b' is also tested.
        //                                                                See:
https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
      * @dev Integer division of two unsigned integers truncating the quotient,
      * reverts on division by zero.
      */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't
hold

        return c;
    }

    /**
```

```
     * @dev Subtracts two unsigned integers, reverts on overflow
     * (i.e. if subtrahend is greater than minuend).
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Adds two unsigned integers, reverts on overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    /**
     * @dev Divides two unsigned integers and returns the remainder (unsigned
integer modulo),
     * reverts when dividing by zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}

/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic
authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
    address public owner;


    event OwnershipTransferred(address indexed previousOwner, address indexed
newOwner);
```

```
    /**
     * @dev The Ownable constructor sets the original `owner` of the contract
to the sender
     * account.
     */
    constructor() public {
        owner = msg.sender;
    }


    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(msg.sender == owner);
        _;
    }


    /**
     * @dev Allows the current owner to transfer control of the contract to a
newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function transferOwnership(address newOwner) onlyOwner public {
        require(newOwner != address(0));
        emit OwnershipTransferred(owner, newOwner);
        owner = newOwner;
    }

}

/**
 * @title Pausable
 * @dev Base contract which allows children to implement an emergency stop
mechanism.
 */
contract Pausable is Ownable {
    event Pause();
    event Unpause();

    bool public paused = false;


    /**
```

```
    * @dev Modifier to make a function callable only when the contract is not
paused.
    */
    modifier whenNotPaused() {
        require(!paused);
        _;
    }

    /**
    * @dev Modifier to make a function callable only when the contract is paused.
    */
    modifier whenPaused() {
        require(paused);
        _;
    }

    /**
    * @dev called by the owner to pause, triggers stopped state
    */
    function pause() public onlyOwner whenNotPaused {
        paused = true;
        emit Pause();
    }

    /**
    * @dev called by the owner to unpause, returns to normal state
    */
    function unpause() public onlyOwner whenPaused {
        paused = false;
        emit Unpause();
    }
}

/**
 * @title Blacklistable Token
 * @dev Allows accounts to be blacklisted by a "blacklister" role
*/
contract Blacklistable is Pausable {

    address public blacklister;
    mapping(address => bool) internal blacklisted;

    event Blacklisted(address indexed _account);
    event UnBlacklisted(address indexed _account);
    event BlacklisterChanged(address indexed newBlacklister);
```

```
constructor() public {
    blacklister = msg.sender;
}

/**
 * @dev Throws if called by any account other than the blacklister
*/
modifier onlyBlacklister() {
    require(msg.sender == blacklister);
    _;
}

/**
 * @dev Throws if argument account is blacklisted
 * @param _account The address to check
*/
modifier notBlacklisted(address _account) {
    require(blacklisted[_account] == false);
    _;
}

/**
 * @dev Checks if account is blacklisted
 * @param _account The address to check
*/
function isBlacklisted(address _account) public view returns (bool) {
    return blacklisted[_account];
}

/**
 * @dev Adds account to blacklist
 * @param _account The address to blacklist
*/
function blacklist(address _account) public onlyBlacklister {
    blacklisted[_account] = true;
    emit Blacklisted(_account);
}

/**
 * @dev Removes account from blacklist
 * @param _account The address to remove from the blacklist
*/
function unBlacklist(address _account) public onlyBlacklister {
    blacklisted[_account] = false;
```

```
        emit UnBlacklisted(_account);
    }

    function updateBlacklister(address _newBlacklister) public onlyOwner {
        require(_newBlacklister != address(0));
        blacklister = _newBlacklister;
        emit BlacklisterChanged(blacklister);
    }
}


contract StandardToken is IERC20, IERC223, Pausable, Blacklistable {
    uint256 public totalSupply;

    using SafeMath for uint;

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

    function balanceOf(address _owner) public view returns (uint256 balance)
{
        return balances[_owner];
    }

    function transferFrom(address _from, address _to, uint256 _value)
whenNotPaused        notBlacklisted(_to)        notBlacklisted(msg.sender)
notBlacklisted(_from) public returns (bool) {
        require(_to != address(0));
        require(_value <= balances[_from]);
        require(_value <= allowed[_from][msg.sender]);
        balances[_from] = balances[_from].sub(_value);
        balances[_to] = balances[_to].add(_value);
        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
        emit Transfer(_from, _to, _value);
        return true;
    }

    function approve(address _spender, uint256 _value) whenNotPaused
notBlacklisted(msg.sender) notBlacklisted(_spender) public returns (bool) {
        allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);
        return true;
    }

    function allowance(address _owner, address _spender) public view returns
```

```
(uint256) {
        return allowed[_owner][_spender];
    }

    function increaseApproval(address _spender, uint _addedValue)
whenNotPaused notBlacklisted(msg.sender) notBlacklisted(_spender) public
returns (bool) {
        allowed[msg.sender][_spender]                                    =
allowed[msg.sender][_spender].add(_addedValue);
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }

    function decreaseApproval(address _spender, uint _subtractedValue)
whenNotPaused notBlacklisted(msg.sender) notBlacklisted(_spender) public
returns (bool) {
        uint oldValue = allowed[msg.sender][_spender];
        if (_subtractedValue > oldValue) {
            allowed[msg.sender][_spender] = 0;
        } else {
            allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
        }
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }

    // Function that is called when a user or another contract wants to transfer
funds.
    function transfer(address _to, uint _value, bytes memory _data)
whenNotPaused notBlacklisted(msg.sender) notBlacklisted(_to) public returns
(bool success) {
        if (isContract(_to)) {
            return transferToContract(_to, _value, _data);
        } else {
            return transferToAddress(_to, _value, _data);
        }
    }

    // Standard function transfer similar to ERC20 transfer with no _data.
    // Added due to backwards compatibility reasons.
    function transfer(address _to, uint _value) whenNotPaused
notBlacklisted(msg.sender) notBlacklisted(_to) public returns (bool success)
{
        bytes memory empty;
        if (isContract(_to)) {
```

```
            return transferToContract(_to, _value, empty);
        } else {
            return transferToAddress(_to, _value, empty);
        }
    }

    // Assemble the given address bytecode. If bytecode exists then the _addr
is a contract.
    function isContract(address _addr) private view returns (bool is_contract)
{
        uint length;
        require(_addr != address(0));
        assembly {
            //retrieve the size of the code on target address, this needs assembly
            length := extcodesize(_addr)
        }
        return (length > 0);
    }

    // Function that is called when transaction target is an address.
    function transferToAddress(address _to, uint _value, bytes memory _data)
private returns (bool success) {
        require(balances[msg.sender] >= _value);
        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        emit Transfer(msg.sender, _to, _value);
        emit Transfer(msg.sender, _to, _value, _data);
        return true;
    }

    // Function that is called when transaction target is a contract.
    function transferToContract(address _to, uint _value, bytes memory _data)
private returns (bool success) {
        require(balances[msg.sender] >= _value);
        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        ContractReceiver receiver = ContractReceiver(_to);
        receiver.tokenFallback(msg.sender, _value, _data);
        emit Transfer(msg.sender, _to, _value);
        emit Transfer(msg.sender, _to, _value, _data);
        return true;
    }
}

/**
```

```
 * @title Mintable token
 * @dev Simple ERC20 Token example, with mintable token creation
 * @dev Issue: * https://github.com/OpenZeppelin/zeppelin-solidity/issues/120
 *         Based         on         code         by         TokenMarketNet:
https://github.com/TokenMarketNet/ico/blob/master/contracts/MintableToken.s
ol
 */
contract MintableToken is StandardToken {
    event Mint(address indexed to, uint256 amount);
    event MintFinished();

    bool public mintingFinished = false;


    modifier canMint() {
        require(!mintingFinished);
        _;
    }

    /**
     * @dev Function to mint tokens
     * @param _to The address that will receive the minted tokens.
     * @param _amount The amount of tokens to mint.
     * @return A boolean that indicates if the operation was successful.
     */
    function mint(address _to, uint256 _amount) onlyOwner canMint public returns
(bool) {
        totalSupply = totalSupply.add(_amount);
        balances[_to] = balances[_to].add(_amount);
        emit Mint(_to, _amount);
        emit Transfer(address(0), _to, _amount);
        return true;
    }

    /**
     * @dev Function to stop minting new tokens.
     * @return True if the operation was successful.
     */
    function finishMinting() onlyOwner canMint public returns (bool) {
        mintingFinished = true;
        emit MintFinished();
        return true;
    }
}
```

```
/**
 * @title Burnable Token
 * @dev Token that can be irreversibly burned (destroyed).
 */
contract BurnableToken is MintableToken {

  event Burn(address indexed burner, uint256 value);

  /**
   * @dev Burns a specific amount of tokens.
   * @param _value The amount of token to be burned.
   */
  function burn(uint256 _value) public {
    _burn(msg.sender, _value);
  }

  function _burn(address _who, uint256 _value) internal {
    require(_value <= balances[_who]);
    // no need to require value <= totalSupply, since that would imply the
    // sender's balance is greater than the totalSupply, which *should* be an
assertion failure

    balances[_who] = balances[_who].sub(_value);
    totalSupply = totalSupply.sub(_value);
    emit Burn(_who, _value);
    emit Transfer(_who, address(0), _value);
  }
}

contract EBASE is BurnableToken {
    string public constant name = "EURBASE Stablecoin";
    string public constant symbol = "EBASE";
    uint8 public constant decimals = 18;
    uint256 public constant initialSupply = 1000000 * 10 ** uint256(decimals);

    constructor () public {
        totalSupply = initialSupply;
        balances[msg.sender] = initialSupply;
    }
}
```