JohnWick Sec

Guard the internet of value

ETH SMART CONTRACT AUDIT REPORT

JOHNWICK SECURITY LAB

WWW.JOHNWICK.IO

John Wick Security Lab received the **LT** (company/team) **LittleBeeX (LT)** project smart contract code audit requirements on **2019/04/16.**

Project Name: LittleBeeX (LT)

Smart Contract Address:

https://etherscan.io/address/0x265e709f99e0a00702658be04198cf0cdeb63eb3#code

Audit Number: 20190406

Audit Date: 20190416

Audit Category and Result:

Main Category	Subcategory	Result
Contract programming	Solidity version not specified	Pass
	Solidity version too old	Pass
	Integer overflow/underflow	Pass
	Function input parameters lack of check	Pass
	Function input parameters check bypass	Pass
	Function access control lacks management	Pass
	Critical operation lacks event log	Pass
	human/contract checks bypass	Pass
	Random number generation/use vulnerability	Pass
	Fallback function misuse	Pass
	Race condition	Pass
	Logical vulnerability	Pass
	Other programming issues	Pass
Code specification	Function visibility not explicitly declared	Pass
	Var. storage location not explicitly declared	Pass
	Use keywords/functions to be deprecated	Pass
	Other code specification issues	Pass
GAS optimization	assert() misuse	Pass
	High consumption `for/while` loop	Pass
	High consumption `storage` storage	Pass
	“Out of Gas” Attack	Pass
Business Risk	Evil mint/burn	Pass
	The maximum limit for mintage not sets	Pass
	"Fake Charge" Attack	Pass
	"Short Address" Attack	Pass
	"Double Spend" Attack	Pass
Auto fuzzing		Pass

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

Audit Result: PASS

Auditor: John Wick Security Lab

Audit Details:

//JohnWick:
This contract uses the SafeMath library to avoid potential integer overflow issues, which is in line with the recommended practice.

//JohnWick: 337L
`approve(address _spender, uint256 _value)` function adds the necessary detection code to avoid the race condition problem.

//JohnWick: 494L
`mint(address _to, uint256 _amount) onlyOwner canMint` function sets the upper limit `cap` for the `totalSupply_`, which is in line with the recommended practice.

//JohnWick: 223L 310L
The transfer/transferFrom functions use `frozenAccount[]` and `frozenTimestamp[]` to implement the account freeze function, which can prevent the potential harm of malicious users.

//JohnWick:
Conclusion: Based on the ERC20 standard, this smart contract adds functions such as burn, mint, pause, freeze, etc. Our smart contract audit

team conducted a rigorous security audit of this contract and found no known security issues. The coding of this smart contract is in line with best security practices.

//JohnWick:
Caveat: Because the transfer and approve/transferFrom functions use the `whenNotPaused` function modifier, this gives the contract owner the power to pause/resume the contract transaction at any time, which requires sufficient attention from the digital currency exchange.

Smart Contract Source Code:

```
pragma solidity ^0.5.7;

/*
#############&&############################################################
######################################################################&
###########@%!!$###########################################################
######################################################################&
##########&!!!!!%@#####################$&##########&$@####################&
|%################$!!!!!!|&############################&|!$#######&|!$$
#########%!!!!!!!&###################@%|&##########$%@#####$$########$$######&
|%################$!%###@|!&##############################$!|@###%!%@#&
#$!%@##&|!!!!!!!!!%###$||&##########@%|&##########&$@###&|!||%@##&|!||%@##&
|%######$||%&#####$!%###@||&######$||%&########$||%&########@%!%%!|&###&
@|!!!!!!!!!!!!!!!!!!!!!!$##########@%|&##########%|&###&||&#####&||&######&
|%####$!$##&|%@###$!!!!!!%@#####$!%##@||@###$!%@#@%|&#########$!!$#####&
&!!!!!!!!!!!!!!!!!!!!!!!%@#########@%|&##########%|&###@%%@#####@%%@#####&
|%###&|!|||||!$###$!%####&|!&##@|!|||||!$##@|!|||||!$#######%!|%!|@###&
$!!!!!!!!!!!!!!!!!!!!!!!!&#########@%|&##########%|&###@%%@#####@%%@#####&
|%###@||@#########$!%#####$!$##@||@########@%|&###########$!|&##@%!%##&
%!!!!!!!!!!!!!!!!!!!!!!!!$#########@%!|||||&###%|&#####$!|%|$###%!|%|$###&
|%####&|!|%%||&###$!!||||!!%@###@|!!%%|!$###@%!!|%|!$###&|!$######&|!&&
!!!!!!!!!!!!!!!!!!!!!!!!!%@##############################################
####################################################################$
$!!!!!!!!!!!!!!!!!!!!!!!!|&##############################################
####################################################################&
###%!!!!!!!!!!!!!!!!!|$#################################################
####################################################################&
#####@%!!!!!!!!!!!!!$##############@$Forever
young:#####################################################################
######################&
########&|!!!!!!!%@################@DEFENG XU/LARRY YE/MR GUO/MR LIU/YIRANG
ZHANG/MINGCONG WU/YIMING WANG/YANPENG ZHANG/GAO JUN/KOREAN FRIENDS#&
```

```
###########$|!!|&##########################################################
######################################################################&
#############&&############################################################
######################################################################&

/*
COPYRIGHT (C) 2019 LITTLEBEEX TECHNOLOGY，ALL RIGHTS RESERVED.

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

/**
 * @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 OwnershipRenounced(address indexed previousOwner);
  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) public onlyOwner {
    require(newOwner != address(0));
    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
  }
}

/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */

library SafeMath {

/**
 * @dev Multiplies two numbers, throws on overflow.
 */

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

/**
* @dev Integer division of two numbers, truncating the quotient.
*/

  function div(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;
  }

/**
* @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater
than minuend).
*/

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

/**
* @dev Adds two numbers, throws on overflow.
*/

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

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() onlyOwner whenNotPaused public {
    paused = true;
    emit Pause();
  }

/**
* @dev called by the owner to unpause, returns to normal state
*/

  function unpause() onlyOwner whenPaused public {
    paused = false;
    emit Unpause();
  }
}

/**
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/179
 */

contract ERC20Basic is Pausable {
  function totalSupply() public view returns (uint256);
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}

/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */

contract ERC20 is ERC20Basic {
  function allowance(address owner, address spender) public view returns
(uint256);
  function transferFrom(address from, address to, uint256 value) public returns
(bool);
  function approve(address spender, uint256 value) public returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}

contract BasicToken is ERC20Basic {
  using SafeMath for uint256;
  mapping (address => bool) public frozenAccount; //Accounts frozen indefinitely
  mapping (address => uint256) public frozenTimestamp; //Limited frozen accounts
  mapping(address => uint256) balances;
  uint256 totalSupply_;

/**
* @dev total number of tokens in existence
*/

  function totalSupply() public view returns (uint256) {
    return totalSupply_;
  }

/**
* @dev transfer token for a specified address
* @param _to The address to transfer to.
* @param _value The amount to be transferred.
*/

  function transfer(address _to, uint256 _value) public returns (bool) {
    require(_to != address(0));
    require(_value <= balances[msg.sender]);
    require(!frozenAccount[msg.sender]);
    require(now > frozenTimestamp[msg.sender]);
    require(!frozenAccount[_to]);
    require(now > frozenTimestamp[_to]);
    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    emit Transfer(msg.sender, _to, _value);
    return true;
  }
```

```
/**
* @dev Gets the balance of the specified address.
* @param _owner The address to query the the balance of.
* @return An uint256 representing the amount owned by the passed address.
*/

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

  /**@dev Lock account */

  function freeze(address _target,bool _freeze) onlyOwner public returns (bool)
{
    require(_target != address(0));
    frozenAccount[_target] = _freeze;
    return true;
  }

  /**@dev Bulk lock account */

  function multiFreeze(address[] memory _targets,bool[] memory _freezes)
onlyOwner public returns (bool) {
    require(_targets.length == _freezes.length);
    uint256 len = _targets.length;
    require(len > 0);
    for (uint256 i = 0; i < len; i= i.add(1)) {
      address _target = _targets[i];
      require(_target != address(0));
      bool _freeze = _freezes[i];
      frozenAccount[_target] = _freeze;
    }
    return true;
  }

  /**@dev Lock accounts through timestamp refer
to:https://www.epochconverter.com */

  function freezeWithTimestamp(address _target,uint256 _timestamp) onlyOwner
public returns (bool) {
    require(_target != address(0));
    frozenTimestamp[_target] = _timestamp;
    return true;
  }
  /**@dev     Batch     lock     accounts     through     timestamp     refer
to:https://www.epochconverter.com */

  function   multiFreezeWithTimestamp(address[]   memory   _targets,uint256[]
memory _timestamps) onlyOwner public returns (bool) {
    require(_targets.length == _timestamps.length);
    uint256 len = _targets.length;
    require(len > 0);
    for (uint256 i = 0; i < len; i = i.add(1)) {
      address _target = _targets[i];
      require(_target != address(0));
      uint256 _timestamp = _timestamps[i];
      frozenTimestamp[_target] = _timestamp;
    }
    return true;
  }
}

/**
 * @title Standard ERC20 token
 * @dev Implementation of the basic standard token.
 * @dev https://github.com/ethereum/EIPs/issues/20
 *       @dev       Based       on       code       by       FirstBlood:
https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBlood
Token.sol
 */

contract StandardToken is ERC20, BasicToken {
  mapping (address => mapping (address => uint256)) internal allowed;

/**
* @dev Transfer tokens from one address to another
* @param _from address The address which you want to send tokens from
* @param _to address The address which you want to transfer to
* @param _value uint256 the amount of tokens to be transferred
*/

  function transferFrom(address _from, address _to, uint256 _value) public
returns (bool) {
    require(_to != address(0));
    require(_value <= balances[_from]);
    require(_value <= allowed[_from][msg.sender]);
    require(!frozenAccount[_from]);
    require(!frozenAccount[_to]);
    require(now > frozenTimestamp[_from]);
    require(now > frozenTimestamp[_to]);
    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;
  }

/**
* @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
*
* 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:
* https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
* @param _spender The address which will spend the funds.
* @param _value The amount of tokens to be spent.
*/

  function approve(address _spender, uint256 _value) public returns (bool) {
    require(_value == 0 || allowed[msg.sender][_spender] == 0);
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
  }

/**
* @dev Function to check the amount of tokens that an owner allowed to a spender.
* @param _owner address The address which owns the funds.
* @param _spender address The address which will spend the funds.
* @return A uint256 specifying the amount of tokens still available for the spender.
*/

  function allowance(address _owner, address _spender) public view returns (uint256) {
    return allowed[_owner][_spender];
  }

/**
* @dev Increase the amount of tokens that an owner allowed to a spender.
```

```
*
* approve should be called when allowed[_spender] == 0. To increment
* allowed value is better to use this function to avoid 2 calls (and wait until
* the first transaction is mined)
* @param _spender The address which will spend the funds.
* @param _addedValue The amount of tokens to increase the allowance by.
*/

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

/**
* @dev Decrease the amount of tokens that an owner allowed to a spender.
*
* approve should be called when allowed[_spender] == 0. To decrement
* allowed value is better to use this function to avoid 2 calls (and wait until
* the first transaction is mined)
* From MonolithDAO Token.sol
* @param _spender The address which will spend the funds.
* @param _subtractedValue The amount of tokens to decrease the allowance by.
*/

  function decreaseApproval(address _spender, uint _subtractedValue) 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;
  }
}

/**
 * @title Burn token
 * @dev Token can be destroyed.
 */
```

```
contract BurnableToken is BasicToken {
  event Burn(address indexed burner, uint256 value);

/**
* @dev Destroy the specified number of token.
* @param _value Number of destroyed token.
*/

  function burn(uint256 _value) public {
    _burn(msg.sender, _value);
  }

  function _burn(address _who, uint256 _value) internal {
    require(_value <= balances[_who]);
    //No need to verify value <= totalSupply，Because this means that the balance
of the sender is greater than the total supply. This should be the assertion
failure.
    balances[_who] = balances[_who].sub(_value);
    totalSupply_ = totalSupply_.sub(_value);
    emit Burn(_who, _value);
    emit Transfer(_who, address(0), _value);
  }
}

/**
 * @title StandardBurn token
 * @dev Add the burnFrom method to the ERC20 implementation.
 */

contract StandardBurnableToken is BurnableToken, StandardToken {

/**
* @dev Destroy a specific number of token from target address and reduce the
allowable amount.
* @param _from address token Owner address
* @param _value uint256 Number of destroyed token
*/

  function burnFrom(address _from, uint256 _value) public {
    require(_value <= allowed[_from][msg.sender]);
    //Should https://github.com/OpenZeppelin/zeppelin-solidity/issues/707 be
accepted,
    //This method requires triggering an event with updated approval.
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
    _burn(_from, _value);
  }
}

contract MintableToken is StandardBurnableToken {
  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) {
    require(_to != address(0));
    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;
  }
}

contract CappedToken is MintableToken {
  uint256 public cap;
  constructor(uint256 _cap) public {
  require(_cap > 0);
  cap = _cap;
  }

/**
* @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) {
    require(totalSupply_.add(_amount) <= cap);
    return super.mint(_to, _amount);
  }
}

contract PausableToken is StandardToken {

  function transfer(address _to, uint256 _value) public whenNotPaused returns
(bool) {
    return super.transfer(_to, _value);
  }

  function transferFrom(address _from, address _to, uint256 _value) public
whenNotPaused returns (bool) {
    return super.transferFrom(_from, _to, _value);
  }

  function approve(address _spender, uint256 _value) public whenNotPaused
returns (bool) {
    return super.approve(_spender, _value);
  }

  function increaseApproval(address _spender, uint _addedValue) public
whenNotPaused returns (bool success) {
    return super.increaseApproval(_spender, _addedValue);
  }

  function decreaseApproval(address _spender, uint _subtractedValue) public
whenNotPaused returns (bool success) {
    return super.decreaseApproval(_spender, _subtractedValue);
  }
}
```

```
contract LT_Token is CappedToken, PausableToken {
  string public constant name = "LittleBeeX"; // solium-disable-line uppercase
  string public constant symbol = "LT"; // solium-disable-line uppercase
  uint8 public constant decimals = 18; // solium-disable-line uppercase
  uint256 public constant INITIAL_SUPPLY = 0;
  uint256  public  constant  MAX_SUPPLY  =  50  *  10000  *  10000  *  (10  **
uint256(decimals));

/**
* @dev Constructor that gives msg.sender all of existing tokens.
*/

  constructor() CappedToken(MAX_SUPPLY) public {
    totalSupply_ = INITIAL_SUPPLY;
    balances[msg.sender] = INITIAL_SUPPLY;
    emit Transfer(address(uint160(0x0)), msg.sender, INITIAL_SUPPLY);
  }

/**
* @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 whenNotPaused
public returns (bool) {
    return super.mint(_to, _amount);
  }

/**
* @dev Function to stop minting new tokens.
* @return True if the operation was successful.
*/

  function finishMinting() onlyOwner canMint whenNotPaused public returns (bool)
{
    return super.finishMinting();
  }

/**@dev Withdrawals from contracts can only be made to Owner.*/

  function withdraw (uint256 _amount) onlyOwner public returns (bool) {
    msg.sender.transfer(_amount);
    return true;
  }

//The fallback function.

  function() payable external {
    revert();
  }
}
```