JohnWick Sec

Guard the internet of value

ETH SMART CONTRACT AUDIT REPORT

JOHNWICK SECURITY LAB

WWW.JOHNWICK.IO

John Wick Security Lab received the **LG** (company/team) **LGame (LG)** project smart contract code audit requirements on **2019/07/13.**

Project Name：LGame (LG)

Smart Contract Address：

https://etherscan.io/address/0x6fe536a1d595c12cbb407c5b2c03999f658a5c72#code

Audit Number：20190704

Audit Date：20190713

Audit Category and Result:

Class	SubClass	Result(Pass/Not Pass)
Code programming	Integer overflow	Pass
	Race condition	Pass
	Logical flaw	Pass
	Denial of service	Pass
	Function parameter check	Pass
	Random number generation	Pass
	Compiler version	Pass
	Hardcoded address	Pass
	ERC20 standard	Pass
Special service	Business risk	Pass
	Contract owner privileges	Pass
	"short address" attack	Pass
	"Fake recharge" attack	Pass
GAS optimization	-	Pass
Automated 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

(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:

//JohnWick:

Compared with the previous audit, the security issues indicated have been fixed in the new smart contract, so this audit did not find any security issues.

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 2019-07-11
*/

pragma solidity ^0.5.0;

library 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 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;
  }
```

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

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

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) public onlyOwner {
    require(newOwner != address(0));
    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
  }

}

contract ERC20Basic {
  uint256 public totalSupply;
  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);
}

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 => uint256) balances;

  /**
  * @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]);

    // SafeMath.sub will throw if there is not enough balance.
    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 balance) {
    return balances[_owner];
  }

}

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]);

    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)
   * From MonolithDAO Token.sol
   * @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;
  }

}

contract RegularToken is StandardToken, Ownable {

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

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

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

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

  function decreaseApproval(address _spender, uint _subtractedValue) public
returns (bool success) {
```

```
    return super.decreaseApproval(_spender, _subtractedValue);
  }
}

contract LGame is RegularToken {
    string public name = "LGame";
    string public symbol = "LG";
    uint public decimals = 18;
    uint public INITIAL_SUPPLY = 21000000000 * (10 ** uint256(decimals));

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