降维安全
JohnWick Sec

守 护 价 值 互 联 网

ETH智能合约
审 计 报 告

降 维 安 全 实 验 室

WWW.JOHNWICK.IO

降维安全实验室于 **2019** 年 **7** 月 **13** 日 收到 **LG**（公司/团队）**LGame (LG)** 项目智能合约源代码安全审计需求。

项目名称：LGame (LG)

合约地址：

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

审计编号：20190704

审计日期：20190713

审计项目及结果：

审计大类	审计子类	审计结果（通过或未通过）
合约编写	整数溢出	通过
	竞争条件	通过
	逻辑漏洞	通过
	拒绝服务	通过
	函数参数检查	通过
	随机数生成使用	通过
	编译器版本	通过
	硬编码地址	通过
	ERC20 标准	通过
特色服务	业务风险	通过
	合约拥有者权限	通过
	“短地址”攻击	通过
	“假充值”攻击	通过
GAS 优化	-	通过
自动化模糊测试	-	通过

(其他未知安全漏洞和以太坊公链设计缺陷不包含在本次审计责任范围内)

审计结果：通过

审计团队：降维安全实验室

（声明：降维安全实验室依据本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于本报告出具后，发生或存在的事实，降维安全实验室无法判断其智能合约安全状况，亦不对此承担任何责任。本报告所做的安全审计分析及其他内容，基于信息提供者截止本报告出具时向降维安全实验室提供的相关材料和文件（简称已提供资料）。降维安全实验室假设：已提供资料不存在缺失、篡改、删减、隐瞒的情况。如已提供资料存在信息缺失、被篡改、被删减、被隐瞒的情况，或资料提供者反应的情况与实际情况不符的，降维安全实验室对由此导致的损失和不利影响不承担任何责任。）

审计详情：

//JohnWick:

跟前次相比,所指出的安全问题在新的智能合约里均已得到修复,本次审计未发现任何安全问题.

注：审计详情中所涉及的代码行号均基于项目方上传于 etherscan.io 的已验证合约源代码,即后附的审计源码.

审计源码：

```
/**
 *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;
    }
}
```