慢雾科技
slow mist

智能合约安全审计报告

慢雾安全团队于 2023-05-09日，收到 AIFloki团队对AIFloki项目智能合约安全审计申请。如下为本次智能合约安全审计细节及结果：

Token 名称：

AIFloki

合约文件名及哈希（SHA256）：

AIFloki.sol

f159cbdd14c9bbd36b91bc2d27cac84a0a9aa677557e371b576046b189a85572

本次审计项及结果：

（其他未知安全漏洞不包含在本次审计责任范围）

序号	审计大类	审计子类	审计结果
1	溢出审计	-	通过
2	条件竞争审计	-	通过
3	权限控制审计	权限漏洞审计	通过
		权限过大审计	通过
4	安全设计审计	Zeppelin 模块使用安全	通过
		编译器版本安全	通过
		硬编码地址安全	通过
		Fallback 函数使用安全	通过
		显现编码安全	通过
		函数返回值安全	通过
		call 调用安全	通过
5	拒绝服务审计	-	通过
6	Gas 优化审计	-	通过
7	设计逻辑审计	-	通过
8	“假充值”漏洞审计	-	通过
9	恶意 Event 事件日志审计	-	通过
10	变量声明及作用域审计	-	通过

11	重放攻击审计	ECDSA 签名重放审计	通过
12	未初始化的存储指针	-	通过
13	算术精度误差	-	通过

备注：审计意见及建议见代码注释 //SlowMist//……

审计结果： 通过

审计编号：0X002305090006

审计日期：2023年5月9日

审计团队：慢雾安全团队

总结：此为代币(token)合约，包含锁仓(tokenVault)部分。合约代币总量可变，Operator 可以通过 burn 函数燃烧任意用户的代币，通过 mint 函数给任意用户无限量铸币，经与项目方沟通，后续会把权限交给跨链桥合约。建议将 changeOperator、changeOwner、changePauser 方法增加事件记录。使用了 OpenZeppelin 的 SafeMath 安全模块，值得称赞的做法。合约不存在溢出、条件竞争问题。

合约源代码如下：

```
/** *Submitted for verification at Etherscan.io on 2017-09-23*/
pragma solidity ^0.4.16;


/** * @title SafeMath * @dev Math operations with safety checks that throw on error */
library SafeMath {
  function mul(uint256 a, uint256 b) internal constant returns (uint256) {
    uint256 c = a * b;
    assert(a == 0 || c / a == b);
    return c;
  }
  function div(uint256 a, uint256 b) internal constant 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 constant returns (uint256) {
    assert(b <= a);
    return a - b;
  }

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

/** * @title ERC20Basic * @dev Simpler version of ERC20 interface * @dev see https://github.com/ethereum/EIPs/issues/179
*/
contract ERC20Basic {
  uint256 public totalSupply;
  function balanceOf(address who) constant returns (uint256);
  function transfer(address to, uint256 value) 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) constant returns (uint256);
  function transferFrom(address from, address to, uint256 value) returns (bool);
  function approve(address spender, uint256 value) returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);}
  contract ERC677 is ERC20 {
  function transferAndCall(address to, uint value, bytes data) returns (bool success);

  event Transfer(address indexed from, address indexed to, uint value, bytes data);}
  contract ERC677Receiver {
  function onTokenTransfer(address _sender, uint _value, bytes _data);}
/** * @title Basic token * @dev Basic version of StandardToken, with no allowances.   */

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) returns (bool) {
    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    Transfer(msg.sender, _to, _value);
    return true;   //SlowMist// 返回值符合 EIP20 规范


  }

  /**   * @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) constant returns (uint256 balance) {
    return balances[_owner];
  }
}

/**  *  @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/FirstBloodToken.sol */
contract StandardToken is ERC20,  BasicToken {

  mapping (address => mapping (address => uint256)) 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) returns (bool) {
    var _allowance = allowed[_from][msg.sender];

    // Check is not needed because sub(_allowance, _value) will already throw if this condition is not met
    // require (_value <= _allowance);

    balances[_from] = balances[_from].sub(_value);
    balances[_to] = balances[_to].add(_value);
    allowed[_from][msg.sender] = _allowance.sub(_value);
    Transfer(_from, _to, _value);
    return true; //SlowMist// 返回值符合 EIP20 规范
  }

  /**    * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.     * @param _spender The address which will spend the funds.     * @param _value The amount of tokens to be spent.     */
  function approve(address _spender, uint256 _value) returns (bool) {
    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);
    return true;   //SlowMist// 返回值符合 EIP20 规范


  }

  /**     * @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) constant returns (uint256 remaining) {
    return allowed[_owner][_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     */
  function increaseApproval (address _spender, uint _addedValue)
    returns (bool success) {
    allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
    Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }

  function decreaseApproval (address _spender, uint _subtractedValue)
    returns (bool success) {
    uint oldValue = allowed[msg.sender][_spender];
    if (_subtractedValue > oldValue) {
      allowed[msg.sender][_spender] = 0;
    } else {
      allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
    }
    Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }
}
contract ERC677Token is ERC677 {

  /**   * @dev transfer token to a contract address with additional data if the recipient is a contact.   * @param _to The address to transfer to.   * @param _value The amount to be transferred.   * @param _data The extra data to be passed to the receiving contract.   */
  function transferAndCall(address _to, uint _value, bytes _data)
    public
    returns (bool success)
  {
    super.transfer(_to, _value);
    Transfer(msg.sender, _to, _value, _data);
    if (isContract(_to)) {
      contractFallback(_to, _value, _data);
    }
    return true;
  }


  // PRIVATE

  function contractFallback(address _to, uint _value, bytes _data)
    private
  {
    ERC677Receiver receiver = ERC677Receiver(_to);
    receiver.onTokenTransfer(msg.sender, _value, _data);
  }

  function isContract(address _addr)
    private
    returns (bool hasCode)
  {
    uint length;
    assembly { length := extcodesize(_addr) }
    return length > 0;
  }
}
contract LinkToken is StandardToken, ERC677Token {
    using SafeMath for uint256;
```

```
uint public totalSupply = 0;
string public constant name = 'ChainLink Token';
uint8 public constant decimals = 18;
string public constant symbol = 'LINK';

address public _owner;
address public _operator;
address public _pauser;
bool public   _ispause;

function LinkToken(address owner, address operator,address pauser)public
{
      _owner = owner;
      _operator = operator;
      _pauser = pauser;
      _ispause = false;
}



modifier onlyOperator(){
      require(msg.sender == _operator);
      _;
}
modifier onlyOwner(){
      require(msg.sender == _owner);
      _;
}

modifier onlyPauser(){
      require(msg.sender == _pauser);
      _;
}

function pause() public onlyPauser{
   _ispause = true;
}
function unpause()public onlyPauser{
   _ispause= false;
}
```

```
//SlowMist// 建议添加事件记录

function changeOperator(address new_operator) public onlyOwner{
      _operator=new_operator;
}


//SlowMist// 建议添加事件记录

function changeOwner(address new_owner) public onlyOwner{
      _owner=new_owner;
}
//SlowMist// 建议添加事件记录

function changePauser(address new_pauser) public onlyOwner{
      _pauser = new_pauser;
}


function _mint(address account, uint256 amount) internal {
   require(amount != 0);
   balances[account] = balances[account].add(amount);
   totalSupply= totalSupply.add(amount);
   Transfer(address(0), account, amount);
}


function _burn(address account, uint256 amount) internal {
   require(amount != 0);
   require(amount <= balances[account]);
   totalSupply = totalSupply.sub(amount);
   balances[account] = balances[account].sub(amount);
   Transfer(account, address(0), amount);
}
//SlowMist// Operator 可以给任意用户无限量铸币

function mint(address to, uint256 value) public onlyOperator onlyUnpause{
        _mint(to,value);
}
//SlowMist// Operator 可以燃烧任意用户 Token

function burn(address to, uint256 value) public onlyOperator onlyUnpause{
        _burn(to, value);
```

```
    }
    /**     * @dev transfer token to a specified address with additional data if the recipient is a contract.     * @param _to The address to transfer to.     * @param _value The amount to be transferred.     * @param _data The extra data to be passed to the receiving contract.     */
    function transferAndCall(address _to, uint _value, bytes _data)
    public
    validRecipient(_to)
    onlyUnpause
    returns (bool success)
    {
    return super.transferAndCall(_to, _value, _data);
    }

    /**     * @dev transfer token to a specified address.     * @param _to The address to transfer to.     * @param _value The amount to be transferred.     */
    function transfer(address _to, uint _value)
    public
    validRecipient(_to)
    onlyUnpause
    returns (bool success)
    {
    return super.transfer(_to, _value);
    }

    /**     * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.     * @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
    validRecipient(_spender)
    returns (bool)
    {
    return super.approve(_spender,   _value);
    }

    /**     * @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
    validRecipient(_to)
    onlyUnpause
    returns (bool)
    {
    return super.transferFrom(_from, _to, _value);
    }



    // MODIFIERS


    modifier validRecipient(address _recipient) {
    require(_recipient != address(0) && _recipient != address(this));
    _;
    }
    modifier onlyUnpause(){
      require(!_ispause);
      _;
    }
}
```