慢雾科技

slow mist

智能合约安全审计报告

慢雾安全团队于 2020-08-25 日，收到 LINA 团队对 LINA 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：

LINA

文件名及 HASH(SHA256)：

LINA.zip

159cfb10ce1b055b2b324357fcf14c4ffed355f0a0ea6b319fe8f5c01396d1b3

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：0X002009010001

审计日期：2020 年 09 月 01 日

审计团队：慢雾安全团队

（**声明：**慢雾仅就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。对于出具以后发生或存在的事实，慢雾无法判断其智能合约安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向慢雾提供的文件和资料（简称“已提供资料”）。慢雾假设：已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，慢雾对由此而导致的损失和不利影响不承担任何责任。慢雾仅对该项目的安全情况进行约定内的安全审计并出具了本报告，慢雾不对该项目背景及其他情况进行负责。）

总结：此为代币(token)合约，不包含锁仓(tokenVault)部分。mint 函数没有设置上限，项目方可以进行任意增发。项目方可随意更改锁仓开始结束时间。项目方可以通过修改 onlyOperator 权限然后调用 setBalanceOf 修改任意账户余额而总量保持不变。使用了 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。综合评估合约无风险。

合约源代码如下：

IERC20.sol

```
pragma solidity >=0.4.24;


interface IERC20 {



    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 (uint);

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

    function allowance(address owner, address spender) external view returns (uint);

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

    function approve(address spender, uint value) external returns (bool);

    function transferFrom(
        address from,
        address to,
        uint value
    ) external returns (bool);

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

    event Approval(address indexed owner, address indexed spender, uint value);
}
```

LinearFinanceToken.sol

```
pragma solidity ^0.5.17;


import "./IERC20.sol";

import "./LnTokenStorage.sol";

import "./LnErc20Handler.sol";


contract LinearFinance is IERC20, LnErc20Handler {



    string public constant TOKEN_NAME = "Linear Finance Token";

    string public constant TOKEN_SYMBOL = "LINA";

    uint8 public constant DECIMALS = 18;


    constructor(

        address payable _proxy,

        LnTokenStorage _tokenStorage,

        address _admin,

        uint _totalSupply

    )

        public

        LnErc20Handler(_proxy, _tokenStorage, TOKEN_NAME, TOKEN_SYMBOL, _totalSupply,

DECIMALS, _admin)
```

```
 {

 }



 //SlowMist// mint 函数没有设置上限，项目方可以进行任意增发

 function _mint(address account, uint256 amount) private  {

     require(account != address(0), "ERC20: mint to the zero address");

     _beforeTokenTransfer(address(0), account, amount);


     tokenStorage.setBalanceOf(account, tokenStorage.balanceOf(account).add(amount));

     totalSupply = totalSupply.add(amount);


     emitTransfer(address(0), account, amount);

 }


 function mint(address account, uint256 amount) external onlyAdmin {

     _mint(account, amount);

 }


function _burn(address account, uint256 amount) private {

     require(account != address(0), "ERC20: burn from the zero address");

     _beforeTokenTransfer(account, address(0), amount);


     tokenStorage.setBalanceOf(account, tokenStorage.balanceOf(account).sub(amount));

     totalSupply = totalSupply.sub(amount);
```

```
    emitTransfer(account, address(0), amount);
}

function _beforeTokenTransfer(address from, address to, uint256 amount) internal {
    super._beforeTokenTransfer(from, to, amount);

    require(!paused, "ERC20Pausable: token transfer while paused");
}

//////////////////////////////////////////////////////////////// paused
bool public paused = false;
modifier notPaused {
    require(!paused, "This action cannot be performed while the contract is paused");
    _;
}

//SlowMist// 在出现重大交易异常时可以暂停所有交易，值得称赞的做法
function setPaused(bool _paused) external onlyAdmin {
    if (_paused == paused) {
        return;
    }
    paused = _paused;
    emit PauseChanged(paused);
```

```
}


//////////////////////////////////////////////////////////

event Staking(address indexed who, uint256 value, uint staketime);
event CancelStaking(address indexed who, uint256 value);
event Claim(address indexed who, uint256 stakeval, uint256 rewardval, uint256 sum);
event PauseChanged(bool isPaused);

struct StakingData {
    uint256 amount;
    uint staketime;
}

address linaToken;
mapping (address => StakingData[]) private stakesdata;
uint private stakingEndTime = 1596805918;
uint private claimStartTime = stakingEndTime + 1 days; // set later
uint256 internal constant MIN_STAKING_AMOUNT = 1e18;

uint256 public stakingRewardFactor = 10;
uint256 public constant stakingRewardDenominator = 100000;

uint256 public accountStakingListLimit = 50;
```

```
function staking(uint256 amount) public notPaused returns (bool) {
    require(block.timestamp < stakingEndTime, "Staking stage has end.");
    require(amount >= MIN_STAKING_AMOUNT, "Staking amount too small.");
    require(stakesdata[msg.sender].length < accountStakingListLimit, "Staking list out of limit.");

    _burn(msg.sender, amount);


    StakingData memory skaking = StakingData({
        amount: amount,
        staketime: block.timestamp
    });
    stakesdata[msg.sender].push(skaking);

    emit Staking(msg.sender, amount, block.timestamp);
    return true;
}


function cancelStaking(uint256 amount) public notPaused returns (bool) {
    require(block.timestamp < stakingEndTime, "Staking stage has end.");
    require(amount > 0, "Invalid amount.");

    uint256 returnToken = amount;
```

```
        StakingData[] storage stakes = stakesdata[msg.sender];
        for (uint256 i = stakes.length; i >= 1 ; i--) {
            StakingData storage lastElement = stakes[i-1];
            if (amount >= lastElement.amount) {
                amount = amount.sub(lastElement.amount);
                stakes.pop();
            } else {
                lastElement.amount = lastElement.amount.sub(amount);
                amount = 0;
            }
            if (amount == 0) break;
        }
        require(amount == 0, "Cancel amount too big then staked.");

        _mint(msg.sender, returnToken);

        emit CancelStaking(msg.sender, returnToken);
        return true;
    }

    function claim() public notPaused returns (bool) {
        require(block.timestamp > claimStartTime, "Too early to claim");
        require(stakingRewardFactor > 0, "Need stakingRewardFactor > 0");
        uint256 total = 0;

        uint256 rewardSum = 0;

        StakingData[] memory stakes = stakesdata[msg.sender];

        require(stakes.length > 0, "Nothing to claim");

        uint256 timesDelta = 1 days;

        for (uint256 i=0; i < stakes.length; i++) {

            uint256 amount = stakes[i].amount;

            total = total.add(amount); // principal


            uint256 stakedays = (claimStartTime.sub(stakes[i].staketime)) / timesDelta;

            uint256 reward =

amount.mul(stakedays).mul(stakingRewardFactor).div(stakingRewardDenominator);

            rewardSum = rewardSum.add(reward);

        }

        delete stakesdata[msg.sender];


        uint256 tomint = total.add(rewardSum);

        _mint(msg.sender, tomint);


        emit Claim(msg.sender, total, rewardSum, tomint);

        return true;

    }
```

```
function set_stakingRewardFactor(uint256 factor) external onlyAdmin() {
    stakingRewardFactor = factor;
}

function rewardFactor() external view returns(uint256, uint256) {
    return (stakingRewardFactor, stakingRewardDenominator);
}
//SlowMist// 项目方可随意更改锁仓开始结束时间,可能导致用户无法按期取回 token
function set_StakingPeriod(uint stakingendtime, uint claimstarttime) external onlyAdmin() {
    require(claimstarttime > stakingendtime);

    stakingEndTime = stakingendtime;
    claimStartTime = claimstarttime;
}

function stakingPeriod() external view returns(uint,uint) {
    return (stakingEndTime, claimStartTime);
}

function stakingBalanceOf(address account) external view returns(uint256) {
    uint256 total = 0;
    StakingData[] memory stakes = stakesdata[account];
    for (uint256 i=0; i < stakes.length; i++) {
            total = total.add(stakes[i].amount);

        }

        return total;

    }

}
```

LnAdmin.sol

```
pragma solidity ^0.5.17;

contract LnAdmin {

    address public admin;

    address public candidate;

    constructor(address _admin) public {

        require(_admin != address(0), "admin address cannot be 0");

        admin = _admin;

        emit AdminChanged(address(0), _admin);

    }

    function setCandidate(address _candidate) external onlyAdmin {

        address old = candidate;

        candidate = _candidate;

        emit candidateChanged( old, candidate);

    }
    function becomeAdmin( ) external {

        require( msg.sender == candidate, "Only candidate can become admin");

        address old = admin;

        admin = candidate;

        emit AdminChanged( old, admin );

    }

    modifier onlyAdmin {

        require( (msg.sender == admin), "Only the contract admin can perform this action");

        _;

    }

    event candidateChanged(address oldCandidate, address newCandidate );

    event AdminChanged(address oldAdmin, address newAdmin);}
```

LnErc20Handler.sol

```
pragma solidity ^0.5.17;


import "./SafeMath.sol";

import "./SafeDecimalMath.sol";


import "./LnAdmin.sol";

import "./LnProxyImpl.sol";
import "./LnTokenStorage.sol";


contract LnErc20Handler is LnAdmin, LnProxyImpl {

    using SafeMath for uint;

    using SafeDecimalMath for uint;


    LnTokenStorage public tokenStorage;


    string public name;

    string public symbol;

    uint public totalSupply;

    uint8 public decimals;


    constructor(

        address payable _proxy,

        LnTokenStorage _tokenStorage,

        string memory _name,

        string memory _symbol,

        uint _totalSupply,

        uint8 _decimals,

        address _admin

    ) public LnAdmin(_admin) LnProxyImpl(_proxy) {

        tokenStorage = _tokenStorage;
        name = _name;

        symbol = _symbol;

        totalSupply = _totalSupply;

        decimals = _decimals;

    }


    function allowance(address owner, address spender) public view returns (uint) {

        return tokenStorage.allowance(owner, spender);

    }


    function balanceOf(address account) external view returns (uint) {

        return tokenStorage.balanceOf(account);

    }


    function setTokenStorage(LnTokenStorage _tokenStorage) external optionalProxy_onlyAdmin {

        tokenStorage = _tokenStorage;

        emitTokenStorageUpdated(address(tokenStorage));

    }



    function _internalTransfer(

        address from,

        address to,
        uint value
    ) internal returns (bool) {

        require(to != address(0) && to != address(this) && to != address(proxy), "Cannot transfer to this address"); //SlowMist// 这类检查很好，避免用户失误导致 Token 转丢
        _beforeTokenTransfer(from, to, value);

        tokenStorage.setBalanceOf(from, tokenStorage.balanceOf(from).sub(value));
        tokenStorage.setBalanceOf(to, tokenStorage.balanceOf(to).add(value));

        emitTransfer(from, to, value);

        return true;
    }

    function _transferByProxy(
        address from,
        address to,
        uint value
    ) internal returns (bool) {
        return _internalTransfer(from, to, value);
    }

    function _transferFromByProxy(
    address sender,
    address from,
    address to,
    uint value
) internal returns (bool) {

    tokenStorage.setAllowance(from, sender, tokenStorage.allowance(from, sender).sub(value));
    return _internalTransfer(from, to, value);
}

function _beforeTokenTransfer(address from, address to, uint256 amount) internal { }

// default transfer
function transfer(address to, uint value) external optionalProxy returns (bool) {
    _transferByProxy(messageSender, to, value);

    return true; //SlowMist// 返回值符合 EIP20 规范
}


// default transferFrom
function transferFrom(
    address from,
    address to,
    uint value
) external optionalProxy   returns (bool) {
    return _transferFromByProxy(messageSender, from, to, value);
}


function approve(address spender, uint value) public optionalProxy returns (bool) {
    address sender = messageSender;

    tokenStorage.setAllowance(sender, spender, value);
    emitApproval(sender, spender, value);
    return true; //SlowMist// 返回值符合 EIP20 规范
}

function addressToBytes32(address input) internal pure returns (bytes32) {
    return bytes32(uint256(uint160(input)));
}

event Transfer(address indexed from, address indexed to, uint value);
bytes32 internal constant TRANSFER_SIG = keccak256("Transfer(address,address,uint256)");

function emitTransfer(
    address from,
    address to,
        uint value

    ) internal {

        proxy.Log3( abi.encode(value),  TRANSFER_SIG, addressToBytes32(from),

addressToBytes32(to) );

    }


    event Approval(address indexed owner, address indexed spender, uint value);

    bytes32 internal constant APPROVAL_SIG = keccak256("Approval(address,address,uint256)");


    function emitApproval(

        address owner,

        address spender,

        uint value

    ) internal {

        proxy.Log3( abi.encode(value),  APPROVAL_SIG, addressToBytes32(owner),

addressToBytes32(spender) );

    }


    event TokenStorageUpdated(address newTokenStorage);

    bytes32 internal constant TOKENSTORAGE_UPDATED_SIG =

keccak256("TokenStorageUpdated(address)");


    function emitTokenStorageUpdated(address newTokenStorage) internal {
        proxy.Log1( abi.encode(newTokenStorage), TOKENSTORAGE_UPDATED_SIG );

    }

}
```

LnOperatorModifier.sol

```
pragma solidity ^0.5.17;


import "./LnAdmin.sol";


contract LnOperatorModifier is LnAdmin {



    address public operator;


    constructor(address _operator) internal {

        require(admin != address(0), "admin must be set");


        operator = _operator;

        emit OperatorUpdated(_operator);

    }


    function setOperator(address _opperator) external onlyAdmin {

        operator = _opperator;

        emit OperatorUpdated(_opperator);

    }
    modifier onlyOperator() {

        require(msg.sender == operator, "Only operator can perform this action");

        _;

    }


    event OperatorUpdated(address operator);

}
```

LnProxyERC20.sol

```
pragma solidity ^0.5.17;

import "./LnProxyImpl.sol";

import "./IERC20.sol";


contract LnProxyERC20 is LnProxyBase, IERC20 {

    constructor(address _admin) public LnProxyBase(_admin) {}


    function name() public view returns (string memory) {



        return IERC20(address(target)).name();

    }


    function symbol() public view returns (string memory) {

```

```
    return IERC20(address(target)).symbol();

}


function decimals() public view returns (uint8) {



    return IERC20(address(target)).decimals();

}


function totalSupply() public view returns (uint256) {



    return IERC20(address(target)).totalSupply();

}


function balanceOf(address account) public view returns (uint256) {



    return IERC20(address(target)).balanceOf(account);

}


function allowance(address owner, address spender) public view returns (uint256) {



    return IERC20(address(target)).allowance(owner, spender);

}


function transfer(address to, uint256 value) public returns (bool) {
    target.setMessageSender(msg.sender);

    IERC20(address(target)).transfer(to, value);

    return true;
}

function approve(address spender, uint256 value) public returns (bool) {

    target.setMessageSender(msg.sender);

    IERC20(address(target)).approve(spender, value);

    return true;
}

function transferFrom(
    address from,
    address to,
    uint256 value
) public returns (bool) {

    target.setMessageSender(msg.sender);
    IERC20(address(target)).transferFrom(from, to, value);

    return true;
}
}
```

LnProxyImpl.sol

```
pragma solidity ^0.5.17;

import "./LnAdmin.sol";

contract LnProxyBase is LnAdmin {
    LnProxyImpl public target;

    constructor(address _admin) public LnAdmin(_admin) {}

    function setTarget(LnProxyImpl _target) external onlyAdmin {
        target = _target;
        emit TargetUpdated(_target);
    }

    function Log0( bytes calldata callData ) external onlyTarget {
        uint size = callData.length;
        bytes memory _callData = callData;

        assembly {

            log0(add(_callData, 32), size)

        }

    }


    function Log1( bytes calldata callData, bytes32 topic1 ) external onlyTarget {

        uint size = callData.length;

        bytes memory _callData = callData;

        assembly {

            log1(add(_callData, 32), size, topic1 )

        }

    }


    function Log2( bytes calldata callData, bytes32 topic1, bytes32 topic2 ) external onlyTarget {

        uint size = callData.length;

        bytes memory _callData = callData;

        assembly {

            log2(add(_callData, 32), size, topic1, topic2 )

        }

    }
    function Log3( bytes calldata callData, bytes32 topic1, bytes32 topic2, bytes32 topic3 ) external onlyTarget {
        uint size = callData.length;
        bytes memory _callData = callData;
        assembly {
            log3(add(_callData, 32), size, topic1, topic2, topic3 )
        }
    }

    function Log4( bytes calldata callData, bytes32 topic1, bytes32 topic2, bytes32 topic3, bytes32 topic4 ) external onlyTarget {
        uint size = callData.length;
        bytes memory _callData = callData;
        assembly {
            log4(add(_callData, 32), size, topic1, topic2, topic3, topic4 )
        }
    }

    function() external payable {


        target.setMessageSender(msg.sender);


        assembly {
            let free_ptr := mload(0x40)
            calldatacopy(free_ptr, 0, calldatasize)

            let result := call(gas, sload(target_slot), callvalue, free_ptr, calldatasize, 0, 0)
            returndatacopy(free_ptr, 0, returndatasize)

            if iszero(result) {
                revert(free_ptr, returndatasize)
            }
            return(free_ptr, returndatasize)
        }
    }

    modifier onlyTarget {
        require(LnProxyImpl(msg.sender) == target, "Must be proxy target");
        _;
    }

    event TargetUpdated(LnProxyImpl newTarget);
}


contract LnProxyImpl is LnAdmin {
```

```
LnProxyBase public proxy;

LnProxyBase public integrationProxy;


address public messageSender;


constructor(address payable _proxy) internal {



    require(admin != address(0), "Admin must be set");


    proxy = LnProxyBase(_proxy);

    emit ProxyUpdated(_proxy);

}


function setProxy(address payable _proxy) external onlyAdmin {

    proxy = LnProxyBase(_proxy);

    emit ProxyUpdated(_proxy);

}


function setIntegrationProxy(address payable _integrationProxy) external onlyAdmin {

    integrationProxy = LnProxyBase(_integrationProxy);

}


function setMessageSender(address sender) external onlyProxy {

    messageSender = sender;
    }

    modifier onlyProxy {
        require(LnProxyBase(msg.sender) == proxy || LnProxyBase(msg.sender) == integrationProxy,
"Only the proxy can call");
        _;
    }

    modifier optionalProxy {
        if (LnProxyBase(msg.sender) != proxy && LnProxyBase(msg.sender) != integrationProxy &&
messageSender != msg.sender) {
            messageSender = msg.sender;
        }
        _;
    }

    modifier optionalProxy_onlyAdmin {
        if (LnProxyBase(msg.sender) != proxy && LnProxyBase(msg.sender) != integrationProxy &&
messageSender != msg.sender) {
            messageSender = msg.sender;
        }
        require(messageSender == admin, "only for admin");
        _;
    }

    event ProxyUpdated(address proxyAddress);
}
```

LnTokenStorage.sol

```
pragma solidity ^0.5.17;

import "./LnAdmin.sol";
import "./LnOperatorModifier.sol";
contract LnTokenStorage is LnAdmin, LnOperatorModifier {
    mapping(address => uint) public balanceOf;
    mapping(address => mapping(address => uint)) public allowance;

    constructor(address _admin, address _operator) public LnAdmin(_admin)
LnOperatorModifier(_operator) {}

    function setAllowance(address tokenOwner, address spender, uint value) external onlyOperator {
        allowance[tokenOwner][spender] = value;
    }
    //SlowMist// 项目方可以通过修改 onlyOperator 权限然后调用 setBalanceOf 修改任意账户余额而
总量保持不变
    function setBalanceOf(address account, uint value) external onlyOperator {
        balanceOf[account] = value;

    }

}
```

SafeDecimalMath.sol

```
pragma solidity ^0.5.17;


import "./SafeMath.sol";

library SafeDecimalMath {

    using SafeMath for uint;


    uint8 public constant decimals = 18;

    uint8 public constant highPrecisionDecimals = 27;


    uint public constant UNIT = 10**uint(decimals);


    uint public constant PRECISE_UNIT = 10**uint(highPrecisionDecimals);

    uint private constant UNIT_TO_HIGH_PRECISION_CONVERSION_FACTOR =
10**uint(highPrecisionDecimals - decimals);


    function unit() external pure returns (uint) {

        return UNIT;

    }
```

```
function preciseUnit() external pure returns (uint) {
    return PRECISE_UNIT;
}

function multiplyDecimal(uint x, uint y) internal pure returns (uint) {

    return x.mul(y) / UNIT;
}

function _multiplyDecimalRound(
    uint x,
    uint y,
    uint precisionUnit
) private pure returns (uint) {
    uint quotientTimesTen = x.mul(y) / (precisionUnit / 10);

    if (quotientTimesTen % 10 >= 5) {
        quotientTimesTen += 10;
    }
    return quotientTimesTen / 10;
}
function multiplyDecimalRoundPrecise(uint x, uint y) internal pure returns (uint) {
    return _multiplyDecimalRound(x, y, PRECISE_UNIT);
```

```
}
function multiplyDecimalRound(uint x, uint y) internal pure returns (uint) {
    return _multiplyDecimalRound(x, y, UNIT);
}
function divideDecimal(uint x, uint y) internal pure returns (uint) {


    return x.mul(UNIT).div(y);
}
function _divideDecimalRound(
    uint x,
    uint y,
    uint precisionUnit
) private pure returns (uint) {
    uint resultTimesTen = x.mul(precisionUnit * 10).div(y);

    if (resultTimesTen % 10 >= 5) {
        resultTimesTen += 10;
    }
    return resultTimesTen / 10;
}
function divideDecimalRound(uint x, uint y) internal pure returns (uint) {
    return _divideDecimalRound(x, y, UNIT);
    }

    function divideDecimalRoundPrecise(uint x, uint y) internal pure returns (uint) {

        return _divideDecimalRound(x, y, PRECISE_UNIT);

    }

    function decimalToPreciseDecimal(uint i) internal pure returns (uint) {

        return i.mul(UNIT_TO_HIGH_PRECISION_CONVERSION_FACTOR);

    }


    function preciseDecimalToDecimal(uint i) internal pure returns (uint) {

        uint quotientTimesTen = i / (UNIT_TO_HIGH_PRECISION_CONVERSION_FACTOR / 10);


        if (quotientTimesTen % 10 >= 5) {

            quotientTimesTen += 10;

        }


        return quotientTimesTen / 10;

    }

}
```

SafeMath.sol

```
pragma solidity >=0.4.24;

//SlowMist// 使用了 OpenZeppelin 的 SafeMath 安全模块，值得称赞的做法

library SafeMath {
```

```
function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");

    return c;
}

function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b <= a, "SafeMath: subtraction overflow");
    uint256 c = a - b;

    return c;
}

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, "SafeMath: multiplication overflow");
```

```
        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, "SafeMath: division by zero");
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0, "SafeMath: modulo by zero");
        return a % b;
    }
}
```

慢雾科技
slow mist

官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号