



智能合约安全审计报告



慢雾安全团队于 2019-04-04 日，收到 OKUSD 团队对 USDK Token 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：

合约名称：

OKUSD

合约文件名及 SHA256 值：

序号	文件名	SHA 256 值
1	Address.sol	dfaab729885b4fcc9b229f0c0a3c84d94e79c1c3452f4db4b99851f726239b6d
2	OKUSDimpl.sol	c984e874243b9c262d3465f7e4c9133da7ec1067fec9987a2f92995367685689
3	OwnedUpgradeabilityProxy.sol	034f5f5450d28bf9852cd1181e40fdd5598ecb8ff3cda43ddb8297c351d7e6b1
4	Proxy.sol	1d11e87a563e72820f113f0646ceb6cafc5506b94a8b080776449123442d59bd
5	SafeMath.sol	49cba5e8c9434328265b435d41c995f3b82337178c66b831c4322690a8f22ddf
6	UpgradeabilityProxy.sol	ee13c4748a1156123e51948eb8e79b000c3d9eda168b7b02cd19dd0b5597b21a

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	未初始化的存储指针	-	通过
11	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：0X001904090001

审计日期：2019 年 04 月 09 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)。合约使用了 OpenZeppelin 的 SafeMath 安全模块，合约不存在溢出、条件竞争问题，值得称赞的做法，合约设计为可升级模型，逻辑合约与数据存储合约分离，可以在不影响正常业务数据的情况下，对逻辑合约进行升级，综合评估合约无风险。

合约源代码如下：

Address.sol

```
pragma solidity ^0.4.24;
```

//SlowMist// 判断目标地址是否为合约地址

```
library Address {
```

```
    /**
```

```
     * Returns whether the target address is a contract
```

```
     * @dev This function will return false if invoked during the constructor of a contract,
```

```
* as the code is not actually created until after the constructor finishes.
* @param account address of the account to check
* @return whether the target address is a contract
*/
function isContract(address account) internal view returns (bool) {
    uint256 size;
    // XXX Currently there is no better way to check if there is a contract in an address
    // than to check the size of the code at that address.
    // See https://ethereum.stackexchange.com/a/14016/36603
    // for more details about how this works.
    // TODO Check this again before the Serenity release, because all addresses will be
    // contracts then.
    // solhint-disable-next-line no-inline-assembly
    assembly { size := extcodesize(account) }
    return size > 0;
}
}
```

OKUSDimpl.sol

//SlowMist// 逻辑合约代码可升级更新

```
pragma solidity ^0.4.24;

import "./SafeMath.sol";

/**
 * @title USDKImplementation
 * @dev this contract is a Pausable ERC20 token with Burn and Mint
 * controleld by a central SupplyController. By implementing USDKImplementation
 * this contract also includes external methods for setting
 * a new implementation contract for the Proxy.
 * NOTE: The storage defined here will actually be held in the Proxy
 * contract and all calls to this contract should be made through
 * the proxy, including admin actions done as owner or supplyController.
 * Any call to transfer against this contract should fail
 * with insufficient funds since no tokens will be issued there.
 */
contract USDKImplementation {
```

```
/**
 * MATH
 */

using SafeMath for uint256;

/**
 * DATA
 */

// INITIALIZATION DATA
bool private initialized = false;

// ERC20 BASIC DATA
mapping(address => uint256) internal balances;
uint256 internal totalSupply_;
string public constant name = "USDK"; // solium-disable-line uppercase
string public constant symbol = "USDK"; // solium-disable-line uppercase
uint8 public constant decimals = 18; // solium-disable-line uppercase

// ERC20 DATA
mapping (address => mapping (address => uint256)) internal _allowed;

// OWNER DATA
address public owner;

// PAUSABILITY DATA
bool public paused = false;

// LAW ENFORCEMENT DATA
address public lawEnforcementRole;
mapping(address => bool) internal frozen;

// SUPPLY CONTROL DATA
address public supplyController;

/**
 * EVENTS
 */

// ERC20 BASIC EVENTS
```

```
event Transfer(address indexed from, address indexed to, uint256 value);
```

```
// ERC20 EVENTS
```

```
event Approval(  
    address indexed owner,  
    address indexed spender,  
    uint256 value  
);
```

```
// OWNABLE EVENTS
```

```
event OwnershipTransferred(  
    address indexed oldOwner,  
    address indexed newOwner  
);
```

```
// PAUSABLE EVENTS
```

```
event Pause();  
event Unpause();
```

```
// LAW ENFORCEMENT EVENTS
```

```
event AddressFrozen(address indexed addr);  
event AddressUnfrozen(address indexed addr);  
event FrozenAddressWiped(address indexed addr);  
event LawEnforcementRoleSet (  
    address indexed oldLawEnforcementRole,  
    address indexed newLawEnforcementRole  
);
```

```
// SUPPLY CONTROL EVENTS
```

```
event SupplyIncreased(address indexed to, uint256 value);  
event SupplyDecreased(address indexed from, uint256 value);  
event SupplyControllerSet(  
    address indexed oldSupplyController,  
    address indexed newSupplyController  
);
```

```
/**
```

```
 * FUNCTIONALITY
```

```
*/
```

```
// INITIALIZATION FUNCTIONALITY
```

```
/**
 * @dev sets 0 initials tokens, the owner, and the supplyController.
 * this serves as the constructor for the proxy but compiles to the
 * memory model of the Implementation contract.
 */
function initialize() public {
    require(!initialized, "already initialized");
    owner = msg.sender;
    lawEnforcementRole = address(0);
    totalSupply_ = 0;
    supplyController = msg.sender;
    initialized = true;
}

/**
 * The constructor is used here to ensure that the implementation
 * contract is initialized. An uncontrolled implementation
 * contract might lead to misleading state
 * for users who accidentally interact with it.
 */
constructor() public {
    initialize();
    pause();
}

// ERC20 BASIC FUNCTIONALITY

/**
 * @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 whenNotPaused returns (bool) {
```

```
require(_to != address(0), "cannot transfer to address zero"); //SlowMist// 这类检查很好,
```

避免用户失误导致 Token 转丢

```
require(!frozen[_to] && !frozen[msg.sender], "address frozen");
require(_value <= balances[msg.sender], "insufficient funds");

balances[msg.sender] = balances[msg.sender].sub(_value);
balances[_to] = balances[_to].add(_value);
emit 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 _addr The address to query the the balance of.
 * @return An uint256 representing the amount owned by the passed address.
 */
function balanceOf(address _addr) public view returns (uint256) {
    return balances[_addr];
}

// ERC20 FUNCTIONALITY

/**
 * @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 whenNotPaused returns (bool)
{

    require(_to != address(0), "cannot transfer to address zero"); //SlowMist// 这类检查很好,
```

避免用户失误导致 Token 转丢

```
require(!frozen[_to] && !frozen[_from] && !frozen[msg.sender], "address frozen");
require(_value <= balances[_from], "insufficient funds");
require(_value <= _allowed[_from][msg.sender], "insufficient allowance");

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; //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.
 * 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 whenNotPaused returns (bool) {
    _approve(msg.sender, spender, value);

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

/**
 * @dev Approve an address to spend another addresses' tokens.
 * @param _owner The address that owns the tokens.
 * @param spender The address that will spend the tokens.
 * @param value The number of tokens that can be spent.
 */
function _approve(address _owner, address spender, uint256 value) internal {
    require(!frozen[spender] && !frozen[_owner], "address frozen");
    require(spender != address(0) && _owner != address(0), "not address(0)");
    _allowed[_owner][spender] = value;
    emit Approval(_owner, spender, value);
}

/**
 * @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[msg.sender][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
 * Emits an Approval event.
 * @param spender The address which will spend the funds.
 * @param addedValue The amount of tokens to increase the allowance by.
 */
function increaseAllowance(address spender, uint256 addedValue) public whenNotPaused returns (bool) {
    _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
    return true;
}

/**
 * @dev Decrease the amount of tokens that an owner allowed to a spender.
 * approve should be called when _allowed[msg.sender][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
 * Emits an Approval event.
 * @param spender The address which will spend the funds.
 * @param subtractedValue The amount of tokens to decrease the allowance by.
 */
function decreaseAllowance(address spender, uint256 subtractedValue) public whenNotPaused returns (bool)
{
    _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue));
    return true;
}

// OWNER FUNCTIONALITY

/**
 * @dev Throws if called by any account other than the owner.
 */
modifier onlyOwner() {
    require(msg.sender == owner, "onlyOwner");
    _;
}
```

```
/**
 * @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 {
```

//SlowMist// 这类检查很好，避免操作失误导致合约控制权丢失

```
    require(_newOwner != address(0), "cannot transfer ownership to address zero");
    emit OwnershipTransferred(owner, _newOwner);
    owner = _newOwner;
}
```

```
// PAUSABILITY FUNCTIONALITY
```

```
/**
 * @dev Modifier to make a function callable only when the contract is not paused.
 */
```

```
modifier whenNotPaused() {
    require(!paused, "whenNotPaused");
    _;
}
```

```
/**
 * @dev called by the owner to pause, triggers stopped state
 */
```

//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 pause() public onlyOwner {
    require(!paused, "already paused");
    paused = true;
    emit Pause();
}
```

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

```
function unpause() public onlyOwner {
    require(paused, "already unpaused");
    paused = false;
    emit Unpause();
}
```

```
// LAW ENFORCEMENT FUNCTIONALITY

/**
 * @dev Sets a new Law enforcement role address.
 * @param _newLawEnforcementRole The new address allowed to freeze/unfreeze addresses and seize their
 tokens.
 */
function setLawEnforcementRole(address _newLawEnforcementRole) public {
    require(msg.sender == lawEnforcementRole || msg.sender == owner, "only lawEnforcementRole or
Owner");

    //SlowMist// 这类检查很好，避免操作失误导致权限设置错误白白消耗 Gas

    require(_newLawEnforcementRole != address(0), "lawEnforcementRole cannot address(0)");
    emit LawEnforcementRoleSet(lawEnforcementRole, _newLawEnforcementRole);
    lawEnforcementRole = _newLawEnforcementRole;
}

modifier onlyLawEnforcementRole() {
    require(msg.sender == lawEnforcementRole, "onlyLawEnforcementRole");
    _;
}

/**
 * @dev Freezes an address balance from being transferred.
 * @param _addr The new address to freeze.
 */

//SlowMist// lawEnforcementRole 角色可以冻结或解冻指定地址

function freeze(address _addr) public onlyLawEnforcementRole {
    require(!frozen[_addr], "address already frozen");
    frozen[_addr] = true;
    emit AddressFrozen(_addr);
}

/**
 * @dev Unfreezes an address balance allowing transfer.
 * @param _addr The new address to unfreeze.
 */

function unfreeze(address _addr) public onlyLawEnforcementRole {
    require(frozen[_addr], "address already unfrozen");
    frozen[_addr] = false;
    emit AddressUnfrozen(_addr);
}
```

```
}

/**
 * @dev Wipes the balance of a frozen address, burning the tokens
 * and setting the approval to zero.
 * @param _addr The new frozen address to wipe.
 */
function wipeFrozenAddress(address _addr) public onlyLawEnforcementRole {
    require(frozen[_addr], "address is not frozen");
    uint256 _balance = balances[_addr];
    balances[_addr] = 0;
    totalSupply_ = totalSupply_.sub(_balance);
    emit FrozenAddressWiped(_addr);
    emit SupplyDecreased(_addr, _balance);
    emit Transfer(_addr, address(0), _balance);
}

/**
 * @dev Gets the balance of the specified address.
 * @param _addr The address to check if frozen.
 * @return A bool representing whether the given address is frozen.
 */
function isFrozen(address _addr) public view returns (bool) {
    return frozen[_addr];
}

// SUPPLY CONTROL FUNCTIONALITY

/**
 * @dev Sets a new supply controller address.
 * @param _newSupplyController The address allowed to burn/mint tokens to control supply.
 */
function setSupplyController(address _newSupplyController) public {
    require(msg.sender == supplyController || msg.sender == owner, "only SupplyController or Owner");

    //SlowMist// 这类检查很好，避免操作失误导致权限设置错误白白消耗 Gas

    require(_newSupplyController != address(0), "cannot set supply controller to address zero");
    emit SupplyControllerSet(supplyController, _newSupplyController);
    supplyController = _newSupplyController;
}

modifier onlySupplyController() {
```

```
        require(msg.sender == supplyController, "onlySupplyController");
        _;
    }

    /**
     * @dev Increases the total supply by minting the specified number of tokens to the supply controller
     account.
     * @param _value The number of tokens to add.
     * @return A boolean that indicates if the operation was successful.
     */
    function increaseSupply(uint256 _value) public onlySupplyController returns (bool success) {
        totalSupply_ = totalSupply_.add(_value);
        balances[supplyController] = balances[supplyController].add(_value);
        emit SupplyIncreased(supplyController, _value);
        emit Transfer(address(0), supplyController, _value);
        return true;
    }

    /**
     * @dev Decreases the total supply by burning the specified number of tokens from the supply controller
     account.
     * @param _value The number of tokens to remove.
     * @return A boolean that indicates if the operation was successful.
     */
    function decreaseSupply(uint256 _value) public onlySupplyController returns (bool success) {
        require(_value <= balances[supplyController], "not enough supply");
        balances[supplyController] = balances[supplyController].sub(_value);
        totalSupply_ = totalSupply_.sub(_value);
        emit SupplyDecreased(supplyController, _value);
        emit Transfer(supplyController, address(0), _value);
        return true;
    }
}
```

OwnedUpgradeabilityProxy.sol

```
pragma solidity ^0.4.24;

import './UpgradeabilityProxy.sol';

/**
```

```
* @title OwnedUpgradeabilityProxy
* @dev This contract combines an upgradeability proxy with basic authorization control functionalities
*/
contract OwnedUpgradeabilityProxy is UpgradeabilityProxy {
    /**
     * @dev Event to show ownership has been transferred
     * @param previousOwner representing the address of the previous owner
     * @param newOwner representing the address of the new owner
     */
    event ProxyOwnershipTransferred(address previousOwner, address newOwner);

    // Storage position of the owner of the contract
    bytes32 private constant proxyOwnerPosition = keccak256("org.zeppelinos.proxy.owner");

    /**
     * @dev the constructor sets the original owner of the contract to the sender account.
     */
    constructor() public {
        setUpgradeabilityOwner(msg.sender);
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyProxyOwner() {
        require(msg.sender == proxyOwner());
        _;
    }

    /**
     * @dev Tells the address of the owner
     * @return the address of the owner
     */
    function proxyOwner() public view returns (address owner) {
        bytes32 position = proxyOwnerPosition;
        assembly {
            owner := sload(position)
        }
    }

    /**
     * @dev Sets the address of the owner

```

```
*/  
function setUpgradeabilityOwner(address newProxyOwner) internal {  
    bytes32 position = proxyOwnerPosition;  
    assembly {  
        sstore(position, newProxyOwner)  
    }  
}  
  
/**  
 * @dev Allows the current owner to transfer control of the contract to a newOwner.  
 * @param newOwner The address to transfer ownership to.  
 */  
function transferProxyOwnership(address newOwner) public onlyProxyOwner {  
    require(newOwner != address(0)); //SlowMist// 这类检查很好，避免操作失误导致权限丢失  
    emit ProxyOwnershipTransferred(proxyOwner(), newOwner);  
    setUpgradeabilityOwner(newOwner);  
}  
  
/**  
 * @dev Allows the proxy owner to upgrade the current version of the proxy.  
 * @param implementation representing the address of the new implementation to be set.  
 */  
  
//SlowMist// 更新 implementation  
  
function upgradeTo(address implementation) public onlyProxyOwner {  
    _upgradeTo(implementation);  
}  
  
/**  
 * @dev Allows the proxy owner to upgrade the current version of the proxy and call the new implementation  
 * to initialize whatever is needed through a low level call.  
 * @param implementation representing the address of the new implementation to be set.  
 * @param data represents the msg.data to be sent in the low level call. This parameter may include the  
function  
 * signature of the implementation to be called with the needed payload  
 */  
function upgradeToAndCall(address implementation, bytes data) payable public onlyProxyOwner {  
    upgradeTo(implementation);  
    require(implementation.delegatecall(data));  
}  
}
```


Proxy.sol

//SlowMist// Proxy 合约用于调用逻辑合约

```
pragma solidity ^0.4.24;

/**
 * @title Proxy
 * @dev Gives the possibility to delegate any call to a foreign implementation.
 */
contract Proxy {
    /**
     * @dev Tells the address of the implementation where every call will be delegated.
     * @return address of the implementation to which it will be delegated
     */
    function implementation() public view returns (address);

    /**
     * @dev Fallback function allowing to perform a delegatecall to the given implementation.
     * This function will return whatever the implementation call returns
     */
    function () payable public {
        address _impl = implementation();
        require(_impl != address(0));

        assembly {
            let ptr := mload(0x40)
            calldatacopy(ptr, 0, calldatasize)
            let result := delegatecall(gas, _impl, ptr, calldatasize, 0, 0)
            let size := returndatasize
            returndatacopy(ptr, 0, size)

            switch result
            case 0 { revert(ptr, size) }
            default { return(ptr, size) }
        }
    }
}
```

SafeMath.sol

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

```
pragma solidity ^0.4.24;

/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {
    /**
     * @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater than minuend).
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Adds two numbers, reverts on overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }
}
```

UpgradeabilityProxy.sol

```
pragma solidity ^0.4.24;

import './Proxy.sol';
import './Address.sol';

/**
 * @title UpgradeabilityProxy
```

```
* @dev This contract represents a proxy where the implementation address to which it will delegate can be upgraded
*/
contract UpgradeabilityProxy is Proxy {
    /**
        * @dev This event will be emitted every time the implementation gets upgraded
        * @param implementation representing the address of the upgraded implementation
        */
    event Upgraded(address indexed implementation);

    // Storage position of the address of the current implementation
    bytes32 private constant implementationPosition = keccak256("org.zeppelinos.proxy.implementation");

    /**
        * @dev Constructor function
        */
    constructor() public {}

    /**
        * @dev Tells the address of the current implementation
        * @return address of the current implementation
        */
    function implementation() public view returns (address impl) {
        bytes32 position = implementationPosition;
        assembly {
            impl := sload(position)
        }
    }

    /**
        * @dev Sets the address of the current implementation
        * @param newImplementation address representing the new implementation to be set
        */
    function setImplementation(address newImplementation) internal {
        require(Address.isContract(newImplementation), "newImplementation is not a contractAddress");
        bytes32 position = implementationPosition;
        assembly {
            sstore(position, newImplementation)
        }
    }

    /**
```

```
* @dev Upgrades the implementation address  
* @param newImplementation representing the address of the new implementation to be set  
*/  
function _upgradeTo(address newImplementation) internal {  
    address currentImplementation = implementation();  
    require(currentImplementation != newImplementation);  
    setImplementation(newImplementation);  
    emit Upgraded(newImplementation);  
}  
}
```



官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号

