Coineer Security

智能合约安全审计报告

Coineer Security 团队于 2019-5-12 日，收到 Self Incentivized Network 团队对 SIN 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

DemeToken

Token 符号

DEMT

合约列表

SafeMath.sol/IERC20.sol/ERC20.sol/ERC20Detailed.sol/DemeToken.sol

合约地址

0xb380284a5eDBb4E52775b59100B39945117e7a0E

链接地址

etherscan.io/address/0xb380284a5eDBb4E52775b59100B39945117e7a0E

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 //Coineer Security//……

审计结果：通过

审计编号：CS201911090002

审计日期：2019 年 11 月 09 日

审计团队：Coineer Security 安全团队

（声明：Coineer Security 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，Coineer Security 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 Coineer Security 提供的文件和资料（简 称“已提供资料”）。Coineer Security 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，Coineer Security 对由此而导致的损失和不利影响不承担任何责任。）

总结：此为代币合约，包含锁仓和销毁功能, 合约使用了 OpenZeppelin 的 SafeMath 安全模块，值得称赞的做法，综合评估合约无风险

链上合约已经过验证，和源代码无误，合约源代码包含了四个文件，SafeMath.sol，IERC20.sol，ERC20.sol，ERC20Detailed.sol.

1. SafeMath.sol

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

```
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;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        uint256 c = a / b;
        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 add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
```

```
    }

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

---

2. IERC20.sol

//Coineer Security// 标准的 ERC20 代币接口，经验证不存在安全漏洞

```
pragma solidity ^0.5.0;

interface IERC20 {
    function transfer(address to, uint256 value) external returns (bool);

    function approve(address spender, uint256 value) external returns (bool);

    function transferFrom(address from, address to, uint256 value) external
returns (bool);

    function totalSupply() external view returns (uint256);

    function balanceOf(address who) external view returns (uint256);

    function allowance(address owner, address spender) external view returns
(uint256);

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

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

---

3. ERC20.sol

//Coineer Security// 使用了 OpenZeppelin 的标准 ERC20 实现，合约不存在溢出，条件竞争问题

```
pragma solidity ^0.5.0;

import "./IERC20.sol";
import "./SafeMath.sol";

contract ERC20 is IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;

    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address owner) public view returns (uint256) {
        return _balances[owner];
    }
```

```
    function allowance(address owner, address spender) public view returns
(uint256) {
        return _allowances[owner][spender];
    }

    function transfer(address to, uint256 value) public returns (bool) {
        _transfer(msg.sender, to, value);
        return true;
    }

    function approve(address spender, uint256 value) public returns (bool) {
        _approve(msg.sender, spender, value);
        return true;
    }

    function transferFrom(address from, address to, uint256 value) public
returns (bool) {
        _transfer(from, to, value);
        _approve(from, msg.sender, _allowances[from][msg.sender].sub(value));
        return true;
    }

    function increaseAllowance(address spender, uint256 addedValue) public
returns (bool) {
        _approve(msg.sender, spender,
_allowances[msg.sender][spender].add(addedValue));
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue)
public returns (bool) {
```

```
        _approve(msg.sender, spender,
_allowances[msg.sender][spender].sub(subtractedValue));
        return true;
    }

    function _transfer(address from, address to, uint256 value) internal {
        require(to != address(0), "ERC20: transfer to the zero address");

        _balances[from] = _balances[from].sub(value);
        _balances[to] = _balances[to].add(value);
        emit Transfer(from, to, value);
    }

    function _mint(address account, uint256 value) internal {
        require(account != address(0), "ERC20: mint to the zero address");

        _totalSupply = _totalSupply.add(value);
        _balances[account] = _balances[account].add(value);
        emit Transfer(address(0), account, value);
    }

    function _burn(address account, uint256 value) internal {
        require(account != address(0), "ERC20: burn from the zero address");

        _totalSupply = _totalSupply.sub(value);
        _balances[account] = _balances[account].sub(value);
        emit Transfer(account, address(0), value);
    }

    function _approve(address owner, address spender, uint256 value) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");
```

```
        _allowances[owner][spender] = value;
        emit Approval(owner, spender, value);
    }

    function _burnFrom(address account, uint256 value) internal {
        _burn(account, value);
        _approve(account, msg.sender,
_allowances[account][msg.sender].sub(value));
    }
}
```

---

4. ERC20Detailed.sol

//Coineer Security// 使用了 OpenZeppelin 的标准 ERC20Detailed 实现，经验证不存在安全漏洞

```
pragma solidity ^0.5.0;

import "./IERC20.sol";

contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    constructor (string memory name, string memory symbol, uint8 decimals)
public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }
```

```
    function name() public view returns (string memory) {
        return _name;
    }

    function symbol() public view returns (string memory) {
        return _symbol;
    }

    function decimals() public view returns (uint8) {
        return _decimals;
    }
}
```

4. ERC20Detailed.sol

//Coineer Security// 初始化代币，总量 100 亿，精度 4 位，没有锁仓，初始 100 亿代币给到参数 owner 指定的字段，在本合约中，owner 值为 0xbc3f9c071123533880fb885113d67a77e9b9e739

```
pragma solidity ^0.5.0;

import "./ERC20.sol";
import "./ERC20Detailed.sol";

contract DemeToken is ERC20, ERC20Detailed {
    uint8 public constant DECIMALS = 4;
    uint256 public constant INITIAL_SUPPLY = 10000000000 * (10 **
uint256(DECIMALS));

    constructor (address owner) public ERC20Detailed("DemeToken", "DEMT",
DECIMALS) {
```

```
        _mint(owner, INITIAL_SUPPLY);
    }
}
```

Coineer Security

联系方式

contact@coineer.me