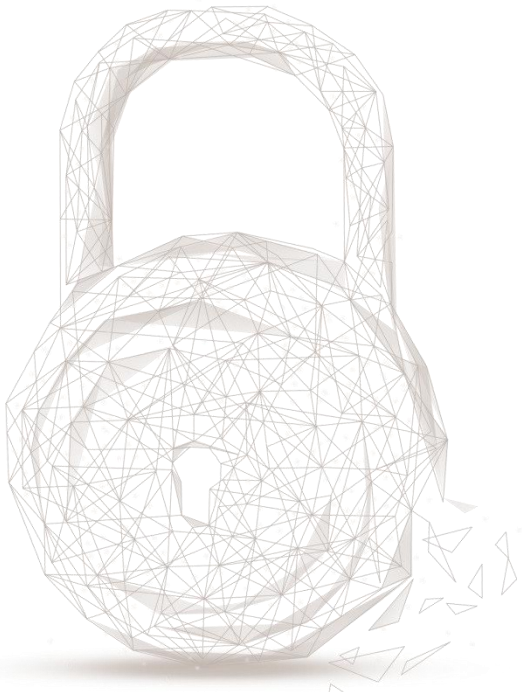
成都链安
BEOSIN

智能合约安全审计报告



审计编号：202101141200

审计项目名称：

Dst

审计项目地址：

无

审计项目链接地址：

无

合约审计开始日期：2021.01.13

合约审计完成日期：2021.01.14

审计结果：通过

审计团队：成都链安科技有限公司

审计类型及结果：

序号	审计类型	审计子项	审计结果
1	代码规范审计	编译器版本安全审计	通过
		弃用项审计	通过
		冗余代码审计	通过
		require/assert 使用审计	通过
		gas 消耗审计	通过
2	通用漏洞审计	整型溢出审计	通过
		重入攻击审计	通过
		伪随机数生成审计	通过
		交易顺序依赖审计	通过
		拒绝服务攻击审计	通过
		函数调用权限审计	通过
		call/delegatecall 安全审计	通过

		返回值安全审计	通过
		tx.origin 使用安全审计	通过
		重放攻击审计	通过
		变量覆盖审计	通过
3	业务审计	业务逻辑审计	通过
		业务实现审计	通过

备注：审计意见及建议请见代码注释。

审计结果说明：

本公司采用形式化验证、静态分析、动态分析、典型案例测试和人工审核的方式对Dst项目智能合约代码规范性、安全性以及业务逻辑三个方面进行多维度全面的安全审计。**经审计，Dst项目智能合约通过所有检测项，合约审计结果为通过。**以下为本合约详细审计信息。

代码规范审计

1. 编译器版本安全审计

老版本的编译器可能会导致各种已知的安全问题，建议开发者在代码中指定合约代码采用最新的编译器版本，并消除编译器告警。

- **安全建议：**无
- **审计结果：通过**

2. 弃用项审计

Solidity智能合约开发语言处于快速迭代中，部分关键字已被新版本的编译器弃用，如throw、years等，为了消除其可能导致的隐患，合约开发者不应该使用当前编译器版本已弃用的关键字。

- **安全建议：无**
- **审计结果：通过**

3. 冗余代码审计

智能合约中的冗余代码会降低代码可读性，并可能需要消耗更多的gas用于合约部署，建议消除冗余代码。

- **安全建议：无**
- **审计结果：通过**

4. require/assert 使用审计

Solidity使用状态恢复异常来处理错误。这种机制将会撤消对当前调用(及其所有子调用)中的状态所做的所有更改，并向调用者标记错误。函数assert和require可用于检查条件并在条件不满足时抛出异常。assert函数只能用于测试内部错误，并检查非变量。require函数用于确认条件有效性，例如输入变量，或合约状态变量是否满足条件，或验证外部合约调用的返回值。

- **安全建议：无**
- **审计结果：通过**

5. gas 消耗审计

以太坊虚拟机执行合约代码需要消耗gas，当gas不足时，代码执行会抛出out of gas异常，并撤销所有状态变更。合约开发者需要控制代码的gas消耗，避免因为gas不足导致函数执行一直失败。

- **安全建议：无**
- **审计结果：通过**

通用漏洞审计

1. 整型溢出审计

整型溢出是很多语言都存在的安全问题，它们在智能合约中尤其危险。Solidity最多能处理256位的数字(2**256-1)，最大数字增加1会溢出得到0。同样，当数字为uint类型时，0减去1会下溢得到最大数字值。溢出情况会导致不正确的结果，特别是如果其可能的结果未被预期，可能会影响程序的可靠性和安全性。

- **安全建议：无**
- **审计结果：通过**

2. 重入攻击审计

重入漏洞是最典型的以太坊智能合约漏洞，曾导致了The DAO被攻击。该漏洞原因是Solidity中的call.value()函数在被用来发送Ether的时候会消耗它接收到的所有gas，当调用call.value()函数发送Ether的逻辑顺序存在错误时，就会存在重入攻击的风险。

- **安全建议：无**
- **审计结果：通过**

3. 伪随机数生成审计

智能合约中可能会使用到随机数，在solidity下常见的是用block区块信息作为随机因子生成，但是这样使用是不安全的，区块信息是可以被矿工控制或被攻击者在交易时获取到，这类随机数在一定程度上是可预测或可碰撞的，比较典型的例子就是fomo3d的airdrop随机数可以被碰撞。

- **安全建议：无**
- **审计结果：通过**

4. 交易顺序依赖审计

在以太坊的交易打包执行过程中，面对相同难度的交易时，矿工往往会选择gas费用高的优先打包，因此用户可以指定更高的gas费用，使自己的交易优先被打包执行。

- **安全建议：无**
- **审计结果：通过**

5. 拒绝服务攻击审计

拒绝服务攻击，即Denial of Service，可以使目标无法提供正常的服务。在以太坊智能合约中也会存在此类问题，由于智能合约的不可更改性，该类攻击可能使得合约永远无法恢复正常工作状态。导致智能合约拒绝服务的原因有很多种，包括在作为交易接收方时的恶意revert、代码设计缺陷导致gas耗尽等等。

- **安全建议：无**
- **审计结果：通过**

6. 函数调用权限审计

智能合约如果存在高权限功能，如：铸币、自毁、change owner等，需要对函数调用做权限限制，避免权限泄露导致的安全问题。

- **安全建议：无**
- **审计结果：通过**

7. call/delegatecall安全审计

Solidity中提供了call/delegatecall函数来进行函数调用，如果使用不当，会造成call注入漏洞，例如call的参数如果可控，则可以控制本合约进行越权操作或调用其他合约的危险函数。

- **安全建议：无**

- **审计结果：通过**

8. 返回值安全审计

在Solidity中存在transfer()、send()、call.value()等方法中，transfer转账失败交易会回滚，而send和call.value转账失败会return false，如果未对返回做正确判断，则可能会执行到未预期的逻辑;另外在ERC20 Token的transfer/transferFrom功能实现中，也要避免转账失败return false的情况，以免造成假充值漏洞。

- **安全建议：无**
- **审计结果：通过**

9. tx.origin使用安全审计

在以太坊智能合约的复杂调用中，tx.origin表示交易的初始创建者地址，如果使用tx.origin进行权限判断，可能会出现错误;另外，如果合约需要判断调用方是否为合约地址时则需要使用tx.origin，不能使用extcodesize。

- **安全建议：无**
- **审计结果：通过**

10. 重放攻击审计

重放攻击是指如果两份合约使用了相同的代码实现，并且身份鉴权在传参中，当用户在向一份合约中执行一笔交易，交易信息可以被复制并且向另一份合约重放执行该笔交易。

- **安全建议：无**
- **审计结果：通过**

11. 变量覆盖审计

以太坊存在着复杂的变量类型，例如结构体、动态数组等，如果使用不当，对其赋值后，可能导致覆盖已有状态变量的值，造成合约执行逻辑异常。

- **安全建议：无**
- **审计结果：通过**

业务审计

1. DstToken代币合约业务功能分析

(1) 代币基本信息

代币名称	DaoSwapToken
代币简称	DST
代币精度	18
代币总量	当前代币总量为0(可铸币，代币上限为所有池子奖励总和)
代币类型	ERC20

表 1 Dst代币基本信息

(2) ERC20代币标准相关的函数

- **业务描述：**DstToken合约所实现的是一个标准的ERC20代币，其相关函数符合ERC20代币标准规范。

  需要注意的是，用户可以使用*approve*函数设置对指定地址的授权值，但为了避免多重授权，建议在需要修改授权值时，不要直接使用*approve*函数进行修改，而是使用*increaseAllowance*和*decreaseAllowance*函数对当前授权值进行增加和减少。
- **相关函数：***name, symbol, decimals, totalSupply, balanceOf, allowance, transfer, transferFrom, approve, increaseAllowance, decreaseAllowance*
- **安全建议：无**
- **审计结果：通过**

(3) 铸币功能和铸币权限管理

- **业务描述：**如下图1，2所示，拥有owner权限的用户或合约可调用*mint*函数向指定地址铸币，当前合约owner为DSTChef合约地址（合约部署后待填），是一个合约地址（owner权限不可转移），代币上限为所有池子奖励总和。

```
contract DSTToken is ERC20("DaoSwapToken", "DST"), Ownable {
    /// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
    function mint(address _to, uint256 _amount) public onlyOwner {
        _mint(_to, _amount);
    }
}
```

图 1 mint函数源码

```
function updatePool(uint256 _pid) public validatePool(_pid) {
    PoolInfo storage pool = poolInfo[_pid];
    if (block.number <= pool.lastRewardBlock) {
        return;
    }
    uint256 lpSupply = pool.lpToken.balanceOf(address(this));
    if (lpSupply == 0) {
        pool.lastRewardBlock = block.number;
        return;
    }
    uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
    uint256 dstReward = multiplier.mul(pool.allocPoint).div(
        totalAllocPoint
    );
    pool.lastRewardBlock = block.number;

    if (dstReward > 0) {
        dst.mint(devaddr, dstReward.div(10)); // 10%
        dst.mint(address(this), dstReward.mul(90).div(100));
        pool.accDstPerShare = pool.accDstPerShare.add(
            dstReward.mul(90).mul(1e12).div(lpSupply).div(100)
        );
    }
}
```

图 2 updatePool函数源码截图

- **相关函数：***mint*、*updatePool*、*balanceOf*、*getMultiplier*、*_mint*
- **安全建议：无**
- **审计结果：通过**

2. DSTChef合约业务功能分析

1. *add*函数

- **业务描述：**如下图1所示，合约实现了*add*函数用于添加pool池，合约所有者可调用该函数添加pool池用于用户抵押获取奖励，要求添加的pool池是首次添加并存储pool池相关信息。

```
function add(
    uint256 _allocPoint,
    IERC20 _lpToken,
    bool _withUpdate
) public onlyOwner {
    if (_withUpdate) {
        massUpdatePools();
    }
    checkPoolDuplicate(_lpToken);
    uint256 lastRewardBlock = block.number > startBlock
        ? block.number
        : startBlock;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    poolInfo.push(
        PoolInfo({
            lpToken: _lpToken,
            allocPoint: _allocPoint,
            lastRewardBlock: lastRewardBlock,
            accDstPerShare: 0
        })
    );
}
```

图 1 add函数源码截图

- **相关函数：***add*、*massUpdatePools*、*checkPoolDuplicate*
- **安全建议：无**
- **审计结果：通过**

2. *set*函数

- **业务描述：**如下图2所示，合约实现了set函数用于设置pool池的奖励分配比例，合约所有者可调用该函数设置pool池的奖励分配比例，pool池奖励分配比例修改后会影响用户提取抵押时的Dst奖励。

```
function set(
    uint256 _pid,
    uint256 _allocPoint,
    bool _withUpdate
) public onlyOwner validatePool(_pid) {
    if (_withUpdate) {
        massUpdatePools();
    }
    totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
        _allocPoint
    );
    poolInfo[_pid].allocPoint = _allocPoint;
}
```

图 2 set函数源码截图

- **相关函数：***set*、*massUpdatePools*
- **安全建议：**建议使用治理合约管理owner权限。
- **审计结果：通过**

3. *getMultiplier*函数

- **业务描述：**如下图3所示，合约实现了*getMultiplier*函数用于计算指定区块_from到_to的Dst奖励，如下图4所示，在设定的startBlock后，第1周Dst奖励为原设定每个区块奖励的3.5倍，第2周Dst奖励为原设定每个区块奖励的2.5倍,第3周Dst奖励为原设定每个区块奖励的1.5倍，第4周Dst奖励为正常设定每个区块奖励，往后每周每个区块奖励在正常设定每个区块奖励的基础上减产20%。

```
function getMultiplier(uint256 _from, uint256 _to)
    public
    view
    returns (uint256)
{
    _from = _from >= startBlock ? _from : startBlock;
    if (_from >= _to) {
        return 0;
    }
    // to always bigger than from
    // from alway >= startBlock
    uint256 weekStart = _from.sub(startBlock).div(blocksPerWeek);
    uint256 weekEnd = _to.sub(startBlock).div(blocksPerWeek);

    // _from and _to in same week.
    if (weekStart == weekEnd) {
        return _to.sub(_from).mul(getMokaPerBlockByWeek(weekStart));
    }

    uint256 dstReward = 0;
    // _from week to block(neex week) reward.
    dstReward = dstReward.add(
        weekStart.add(1).mul(blocksPerWeek).add(startBlock).sub(_from).mul(
            getMokaPerBlockByWeek(weekStart)
        )
    );
    // block(last week of _to) to _to block reward.
    dstReward = dstReward.add(
        _to.sub(weekEnd.mul(blocksPerWeek).add(startBlock)).mul(
            getMokaPerBlockByWeek(weekEnd)
        )
    );

    // (_from week + 1) to (_to week - 1) reward.
    for (
        uint256 weekIndex = weekStart.add(1);
        weekIndex <= weekEnd.sub(1);
        weekIndex++
    ) {
        dstReward = dstReward.add(
            blocksPerWeek.mul(getMokaPerBlockByWeek(weekIndex))
        );
    }

    return dstReward;
```

图 3 getMultiplier函数源码截图

```
function getMokaPerBlockByWeek(uint256 weekIndex)
    public
    view
    returns (uint256)
{
    if (weekIndex == 0) {
        return 700 * (10**18);
    }

    if (weekIndex == 1) {
        return 500 * (10**18);
    }

    if (weekIndex == 2) {
        return 300 * (10**18);
    }

    if (weekIndex == 3) {
        return dstPerBlock;
    }

    uint256 exp = weekIndex - 3;
    // 80% of last week
    return dstPerBlock.mul(8**(exp)).div(10**(exp));
```

图 4 getMokaPerBlockByWeek函数源码截图

- **相关函数：***getMultiplier*、*getMokaPerBlockByWeek*
- **安全建议：无**
- **审计结果：通过**

4. *updatePool*函数

- **业务描述：**如下图5，6所示，合约实现了*updatePool*函数用于更新当前区块pool池的Dst奖励和信息，任意用户可调用该函数更新pool池最新Dst奖励和信息，并调用*mint*函数将距离上次更新新产生的Dst奖励的90%铸至本合约地址，百分之10铸至项目方指定地址（Dst抵押产生的奖励用户得90%，另外10%发送至项目方指定地址，DstToken合约的owner必须为本合约地址）。

```
// Update reward variables of the given pool to be up-to-date.
function updatePool(uint256 _pid) public validatePool(_pid) {
    PoolInfo storage pool = poolInfo[_pid];
    if (block.number <= pool.lastRewardBlock) {
        return;
    }
    uint256 lpSupply = pool.lpToken.balanceOf(address(this));
    if (lpSupply == 0) {
        pool.lastRewardBlock = block.number;
        return;
    }
    uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
    uint256 dstReward = multiplier.mul(pool.allocPoint).div(
        totalAllocPoint
    );
    pool.lastRewardBlock = block.number;

    if (dstReward > 0) {
        dst.mint(devaddr, dstReward.div(10)); // 10%
        dst.mint(address(this), dstReward.mul(90).div(100));
        pool.accDstPerShare = pool.accDstPerShare.add(
            dstReward.mul(90).mul(1e12).div(lpSupply).div(100)
        );
    }
}
```

图 5 updatePool函数源码截图

```
// DSTToken with Governance.
contract DSTToken is ERC20("DaoSwapToken", "DST"), Ownable {
    /// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
    function mint(address _to, uint256 _amount) public onlyOwner {
        _mint(_to, _amount);
    }
}
```

图 6 mint函数源码截图

- **相关函数：***updatePool*、*getMultiplier*、*mint*
- **安全建议：无**
- **审计结果：通过**

5. *deposit*函数

- **业务描述：**如下图7所示，合约实现了*deposit*函数用于用户抵押代币，用户预先授权本合约地址后调用该函数抵押代币（要求用户抵押的pool池存在并且抵押代币数量大于0）。用户抵押时更新pool池信息，本合约代理用户将抵押代币转至合约自身地址。

```
function deposit(uint256 _pid, uint256 _amount) public validatePool(_pid) {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    updatePool(_pid);
    // pending = 0 if user.amount = 0.
    uint256 pending = user.amount.mul(pool.accDstPerShare).div(1e12).sub(
        user.rewardDebt
    );
    user.amount = user.amount.add(_amount);
    user.rewardDebt = user.amount.mul(pool.accDstPerShare).div(1e12);

    if (pending > 0) {
        safeDstTransfer(msg.sender, pending);
        emit WithdrawIncome(msg.sender, _pid, pending);
    }
    if (_amount > 0) {
        pool.lpToken.safeTransferFrom(
            address(msg.sender),
            address(this),
            _amount
        );
        emit Deposit(msg.sender, _pid, _amount);
    }
}
```

图 7 deposit函数源码截图

- **相关函数：***deposit*、*updatePool*、*safeDstTransfer*、*safeTransferFrom*
- **安全建议：无**
- **审计结果：通过**

6. *withdraw*函数

- **业务描述：**如下图8所示，合约实现了*withdraw*函数用于用户提取抵押代币和Dst奖励，用户可调用该函数提取指定pool池指定数量的抵押代币和Dst奖励（要求指定pool池存在并且抵押代币数量大于等于提取数量）。用户提取抵押代币和奖励时更新pool池信息，并将指定的抵押代币和Dst转至用户地址。

```
function withdraw(uint256 _pid, uint256 _amount) public validatePool(_pid) {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    require(user.amount >= _amount, "withdraw: not good");
    updatePool(_pid);

    uint256 pending = user.amount.mul(pool.accDstPerShare).div(1e12).sub(
        user.rewardDebt
    );
    user.amount = user.amount.sub(_amount);
    user.rewardDebt = user.amount.mul(pool.accDstPerShare).div(1e12);

    if (pending > 0) {
        safeDstTransfer(msg.sender, pending);
        emit WithdrawIncome(msg.sender, _pid, pending);
    }
    if (_amount > 0) {
        pool.lpToken.safeTransfer(address(msg.sender), _amount);
        emit Withdraw(msg.sender, _pid, _amount);
    }
}
```

图 8 withdraw函数源码截图

- **相关函数：***withdraw*、*safeDstTransfer*、*safeTransfer*
- **安全建议：无**
- **审计结果：通过**

7. *emergencyWithdraw*函数

- **业务描述：**如下图9所示，合约实现了*emergencyWithdraw*函数用于用户紧急提取抵押代币，用户可调用该函数提取指定pool池全部抵押代币（要求指定pool池存在并且抵押代币数量大于0）。用户调用该函数紧急提取抵押代币时无法获得Dst奖励。

```
function emergencyWithdraw(uint256 _pid) public validatePool(_pid) {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];

    uint256 _amount = user.amount;
    user.amount = 0;
    user.rewardDebt = 0;

    if (_amount > 0) {
        pool.lpToken.safeTransfer(address(msg.sender), _amount);
        emit EmergencyWithdraw(msg.sender, _pid, _amount);
    }
}
```

图 9 emergencyWithdraw函数源码截图

- **相关函数：***emergencyWithdraw*、*safeTransfer*
- **安全建议：无**
- **审计结果：通过**

结论：

Beosin(成都链安)对Dst项目合约的设计和代码实现进行了详细的审计，所有审计过程中发现的问题告知项目方后均已修复。Dst项目审计的总体结果是通过。

成都链安

BEOSIN

官方网址

https://lianantech.com

电子邮箱

vaas@lianantech.com

微信公众号