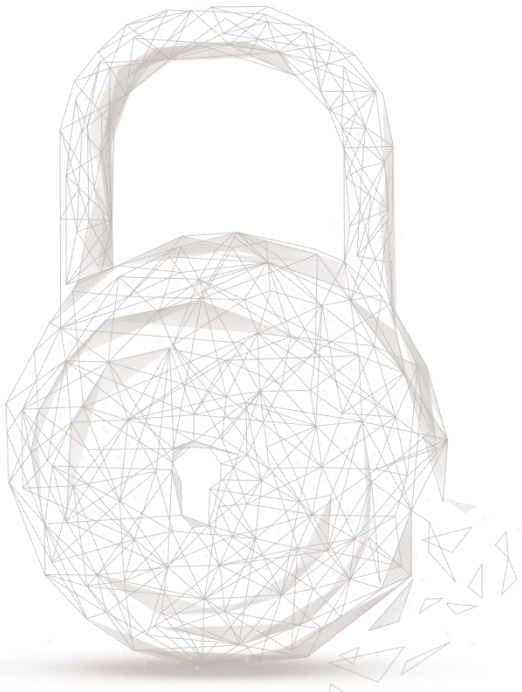
成都链安
BEOSIN

智能合约安全审计报告



审计编号：202009131815

报告查询名称：dUSDTPool

审计合约名称：

dUSDTPool

审计合约地址：

0xe1C00a8C32D6944f61277EA5B8146E5157c2531A

审计合约链接地址：

https://etherscan.io/address/0xe1c00a8c32d6944f61277ea5b8146e5157c2531a#code

合约审计开始日期：2020.09.11

合约审计完成日期：2020.09.13

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dUSDTPool流动性挖矿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，dUSDTPool流动性挖矿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，该漏洞原因是Solidity中的call.value()函数在被用来发送ETH的时候会消耗它接收到的所有gas，当调用call.value()函数发送ETH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. 抵押初始化

- **业务描述：**如下图1，2所示，合约的“抵押-奖励”模式需要初始化相关参数（奖励比例rewardRate、首次更新时间lastUpdateTime、阶段完成时间periodFinish），通过指定的奖励分配管理员地址rewardDistribution调用notifyRewardAmount函数，输入初始用于计算奖励比例的奖励数值reward，初始化抵押与奖励相关参数。

```
contract dUSDTPool is LPTokenWrapper, IRewardDistributionRecipient {
    IERC20 public syfi = IERC20(0xdc38a4846d811572452cB4CE747dc9F5F509820f);
    uint256 public constant DURATION = 7 days;

    uint256 public initreward = 10000*1e18;
    uint256 public starttime = 1599321600; //utc+8 2020 09-06 00:00:00
    uint256 public periodFinish = 0;
    uint256 public rewardRate = 0;
    uint256 public lastUpdateTime;
    uint256 public rewardPerTokenStored;
    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) public rewards;
```

图 1 相关参数代码截图

```
    function notifyRewardAmount(uint256 reward)
        external
        onlyRewardDistribution
        updateReward(address(0))
    {
        if (block.timestamp >= periodFinish) {
            rewardRate = reward.div(DURATION);
        } else {
            uint256 remaining = periodFinish.sub(block.timestamp);
            uint256 leftover = remaining.mul(rewardRate);
            rewardRate = reward.add(leftover).div(DURATION);
        }
        syfi.mint(address(this),reward);
        lastUpdateTime = block.timestamp;
        if(block.timestamp < starttime){
            periodFinish = starttime.add(DURATION);
        }else {
            periodFinish = block.timestamp.add(DURATION);
        }
        emit RewardAdded(reward);
    }
}
```

图 2 notifyRewardAmount函数源码截图

- **相关函数：**notifyRewardAmount、rewardPerToken、lastTimeRewardApplicable
- **安全建议：**无
- **审计结果：**通过

2. **抵押代币**

- **业务描述：**如下图3所示，合约实现了stake函数用于抵押USDT代币，用户预先授权该合约地址，通过调用USDT合约中的transferFrom函数，合约地址代理用户将指定数量的USDT代币转至本合约地址；该函数限制用户仅可在“抵押-奖励”模式开启（到达指定时间）后进行调用；每次调用该函数抵押代币时通过修饰器updateReward更新奖励相关数据；以及每次调用通过修饰器checkhalve检查是否到达阶段完成时间，并进行奖励减半操作和奖励比例与阶段完成时间的更新。

```
function stake(uint256 amount) public updateReward(msg.sender) checkhalve checkStart {
    require(amount > 0, "Cannot stake 0");
    super.stake(amount);
    emit Staked(msg.sender, amount);
}
```

图 3 stake函数源码截图

- **相关函数：** stake、safeTransferFrom、rewardPerToken、lastTimeRewardApplicable、earned、balanceOf
- **安全建议：**无
- **审计结果：**通过

3. 提取抵押代币

- **业务描述：**如下图4所示，合约实现了withdraw函数用于提取已抵押的USDT代币，通过调用USDT合约中的transfer函数，合约地址将指定数量的USDT代币转至函数调用者（用户）地址；该函数限制用户仅可在“抵押-奖励”模式开启（到达指定时间）后进行调用；每次调用该函数抵押代币时通过修饰器updateReward更新奖励相关数据；以及每次调用通过修饰器checkhalve检查是否到达阶段完成时间，并进行奖励减半操作和奖励比例与阶段完成时间的更新。

```
function withdraw(uint256 amount) public updateReward(msg.sender) checkhalve checkStart {
    require(amount > 0, "Cannot withdraw 0");
    super.withdraw(amount);
    emit Withdrawn(msg.sender, amount);
}
```

图 4 withdraw函数源码截图

- **相关函数：**withdraw、safeTransfer、rewardPerToken、lastTimeRewardApplicable、earned、balanceOf
- **安全建议：**无
- **审计结果：**通过

4. 领取抵押奖励

- **业务描述：**如下图5所示，合约实现了getReward函数用于领取抵押奖励（SYFI代币），通过调用syfi合约中的transfer函数，合约地址将指定数量（用户的全部抵押奖励）的SYFI代币转至函数调用者（用户）地址；该函数限制用户仅可在“抵押-奖励”模式开启（到达指定时间）后进行调用；每次调用该函数抵押代币时通过修饰器updateReward更新奖励相关数据；以及每次调用通过修饰器checkhalve检查是否到达阶段完成时间，并进行奖励减半操作和奖励比例与阶段完成时间的更新。

```
function getReward() public updateReward(msg.sender) checkhalve checkStart {
    uint256 reward = earned(msg.sender);
    if (reward > 0) {
        rewards[msg.sender] = 0;
        syfi.safeTransfer(msg.sender, reward);
        emit RewardPaid(msg.sender, reward);
    }
}
```

图 5 getReward函数源码截图

- **相关函数：** getReward、safeTransfer、rewardPerToken、lastTimeRewardApplicable、earned、balanceOf
- **安全建议：**无
- **审计结果：**通过

5. **退出抵押奖励参与**

- **业务描述：**如下图6所示，合约实现了exit函数用于调用者退出抵押奖励参与，调用withdraw函数提取全部已抵押的USDT代币，调用getReward函数领取完调用者的抵押奖励，结束“抵押-奖励”模式参与。

```
function exit() external {
    withdraw(balanceOf(msg.sender));
    getReward();
}
```

图 6 exit函数源码截图

- **相关函数：** exit、withdraw、safeTransfer、getReward、rewardPerToken、lastTimeRewardApplicable、earned、balanceOf
- **安全建议：**无
- **审计结果：**通过

6. **奖励相关数据查询功能**

- **业务描述：**合约用户可通过调用lastTimeRewardApplicable函数查询当前时间戳与阶段完成时间中最早的时间戳；调用rewardPerToken函数可查询每个抵押代币可获得的抵押奖励；调用earned函数可查询指定地址所获取的总抵押奖励。

```
function lastTimeRewardApplicable() public view returns (uint256) {
    return Math.min(block.timestamp, periodFinish);
}

function rewardPerToken() public view returns (uint256) {
    if (totalSupply() == 0) {
        return rewardPerTokenStored;
    }
    return
        rewardPerTokenStored.add(
            lastTimeRewardApplicable()
                .sub(lastUpdateTime)
                .mul(rewardRate)
                .mul(1e18)
                .div(totalSupply())
        );
}

function earned(address account) public view returns (uint256) {
    return
        balanceOf(account)
            .mul(rewardPerToken().sub(userRewardPerTokenPaid[account]))
            .div(1e18)
            .add(rewards[account]);
}
```

图 7 相关函数源码截图

- **相关函数：**lastTimeRewardApplicable、rewardPerToken、earned
- **安全建议：**无
- **审计结果：**通过

结论

Beosin(成都链安)对 dUSDTPool 合约的设计和代码实现进行了详细的审计，奖励代币 SYFI 合约的铸币功能可实现无上限铸币。governance 权限地址（初始为合约部署者）可以添加铸币者地址，拥有铸币权限的地址可以调用 mint 函数无限铸币，影响普通用户的代币兑换，建议项目方妥善使用和添加铸币权限，dUSDTPool 合约审计的总体结果是**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