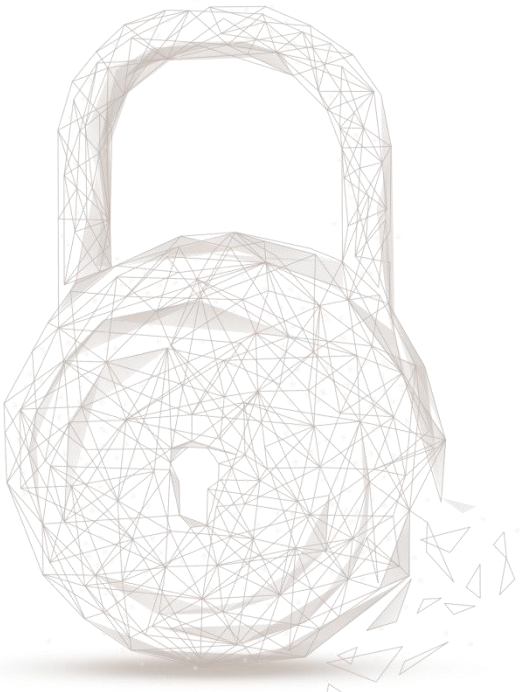
成都链安
BEOSIN

智能合约安全审计报告



成都链安 BEOSIN

审计编号：202103191700

报告查询名称：StableCash

审计合约信息：

合约名称	审计合约地址	审计合约链接地址
STCUSDTLPTokenSharePool	部署后填写	部署后填写
STSUSDTLPTokenSharePool	部署后填写	部署后填写
InitialShareDistributor	部署后填写	部署后填写
Operator	部署后填写	部署后填写
Boardroom	部署后填写	部署后填写
BoardroomRank	部署后填写	部署后填写
Bond	部署后填写	部署后填写
Cash	部署后填写	部署后填写
Distributor	部署后填写	部署后填写
MiningPool	部署后填写	部署后填写
Oracle	部署后填写	部署后填写
ReferRank	部署后填写	部署后填写
Referrer	部署后填写	部署后填写
Share	部署后填写	部署后填写
SimpleERCFund	部署后填写	部署后填写
Timelock	部署后填写	部署后填写
Treasury	部署后填写	部署后填写
VoteProxy	部署后填写	部署后填写

审计文件 Hash（SHA256）：

cf5ce40d67d7c8a79f12a10928aef1152a66ad85b265012ea62dd11f068cbb1c

合约审计开始日期：2021.03.11

合约审计完成日期：2021.03.19

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

科技作出的本审计报告仍存在无法完整检测出全部风险的可能性，成都链安科技对由此产生的损失不承担任何责任。

本声明最终解释权归成都链安科技所有。

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StableCash项目合约代码规范性、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，StableCash项目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。

- **修改建议：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。

- **修改建议：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。

(1) Referrer合约中的未使用的内部函数

```
36    function notifyReferReward(address _referrer, address _referree, uint amount) internal {
37        referReward[_referrer] = referReward[_referrer].add(amount);
38        emit ReferReward(_referrer, _referree, amount);
39    }
```

图 1 notifyReferReward函数源码

- **修改建议：**建议删除合约中冗余代码。
- **修复结果：**项目方已忽略。
- **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。

- **修改建议：无**
- **审计结果：通过**

5. gas 消耗审计

火币生态链Heco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。

- **修改建议：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。

- **修改建议：无**
- **审计结果：通过**

2. 重入攻击审计

重入漏洞原因是Solidity中的call.value()函数在被用来发送HT的时候会消耗它接收到的所有gas，当调用call.value()函数发送HT的逻辑顺序存在错误时，就会存在重入攻击的风险。

- **修改建议：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。

- **修改建议：无**
- **审计结果：通过**

4. 交易顺序依赖审计

在火币生态链Heco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。

- **修改建议：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。在火币生态链Heco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。

- **修改建议：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。

- **修改建议：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。

- **修改建议：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HRC-20 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。

- **修改建议：无**
- **审计结果：通过**

9. tx.origin使用安全审计

在火币生态链Heco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。

- **修改建议：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。

- **修改建议：无**
- **审计结果：通过**

11. 变量覆盖审计

Solidity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。

- **修改建议：无**
- **审计结果：通过**

业务审计

1. Bond合约

(1) 代币基本信息

代币名称	STB
代币简称	STB
代币精度	18
代币总量	初始供应量为0（可铸币，无铸币上限，可销毁）
代币类型	HRC-20

表 1 代币基本信息

(2) 铸币/销毁函数

- **业务描述：**如下图所示，STB代币合约实现了mint、burn和burnFrom函数用于铸币和代币销毁，只有具有operator权限的地址才能调用这些函数进行铸币和代币销毁操作。STB代币铸币无上限，初始铸币数量为0。

```
18    function mint(address recipient_, uint256 amount_)
19        public
20        onlyOperator
21        returns (bool)
22    {
23        uint256 balanceBefore = balanceOf(recipient_);
24        _mint(recipient_, amount_);
25        uint256 balanceAfter = balanceOf(recipient_);
26
27        return balanceAfter > balanceBefore;
28    }
```

图 2 mint函数源码

```
30    function burn(uint256 amount) public override onlyOperator {
31        super.burn(amount);
32    }
33
34    function burnFrom(address account, uint256 amount)
35        public
36        override
37        onlyOperator
38    {
39        super.burnFrom(account, amount);
40    }
```

图 3 burn和burnFrom函数源码

- **相关函数：**mint、onlyOperator、balanceOf、burn、burnFrom
- **审计结果：**通过

2. Cash合约

(1) 代币基本信息

代币名称	STC
代币简称	STC
代币精度	18
代币总量	初始供应量为1（可铸币，无铸币上限，可销毁）
代币类型	HRC-20

表 2 代币基本信息

(2) 铸币/销毁函数

- **业务描述：**如下图所示，STC代币合约实现了mint、burn和burnFrom函数用于铸币和代币销毁，只有具有operator权限的地址才能调用这些函数进行铸币和代币销毁操作。STC代币铸币无上限，初始铸币数量为1枚。

```
30  function mint(address recipient_, uint256 amount_)
31      public
32      onlyOperator
33      returns (bool)
34  {
35      uint256 balanceBefore = balanceOf(recipient_);
36      _mint(recipient_, amount_);
37      uint256 balanceAfter = balanceOf(recipient_);
38
39      return balanceAfter > balanceBefore;
40  }
```

图 4 mint函数源码

```
42  function burn(uint256 amount) public override onlyOperator {
43      super.burn(amount);
44  }
45
46  function burnFrom(address account, uint256 amount)
47      public
48      override
49      onlyOperator
50  {
51      super.burnFrom(account, amount);
52  }
```

图 5 burn和burnFrom函数源码

- **相关函数：**mint、onlyOperator、balanceOf、burn、burnFrom
- **审计结果：**通过

3. Share合约

(1) 代币基本信息

代币名称	STS
代币简称	STS
代币精度	18
代币总量	初始供应量为1（可铸币，无铸币上

	限，可销毁）
代币类型	HRC-20

表 3 代币基本信息

(2) 铸币/销毁函数

- **业务描述：**如下图所示，STS代币合约实现了mint、burn和burnFrom函数用于铸币和代币销毁，只有具有operator权限的地址才能调用这些函数进行铸币和代币销毁操作。STS代币铸币无上限，初始铸币数量为1。

```
30  function mint(address recipient_, uint256 amount_)
31      public
32      onlyOperator
33      returns (bool)
34  {
35      uint256 balanceBefore = balanceOf(recipient_);
36      _mint(recipient_, amount_);
37      uint256 balanceAfter = balanceOf(recipient_);
38
39      return balanceAfter > balanceBefore;
40  }
```

图 6 mint函数源码

```
42  function burn(uint256 amount) public override onlyOperator {
43      super.burn(amount);
44  }
45
46  function burnFrom(address account, uint256 amount)
47      public
48      override
49      onlyOperator
50  {
51      super.burnFrom(account, amount);
52  }
```

图 7 burn和burnFrom函数源码

- **相关函数：**mint、onlyOperator、balanceOf、burn、burnFrom
- **审计结果：**通过

4. BoardroomRank合约

(1) 代币基本信息

代币名称	BoardroomRank
代币简称	BRR

代币精度	18
代币总量	初始供应量为0（可铸币，无铸币上限，可销毁）
代币类型	HRC-20

表 4 代币基本信息

(2) 铸币/销毁/转账函数

- **业务描述：**如下图所示，BRR代币合约实现了mint和burnFrom函数用于铸币和代币销毁，以及transfer和burn函数用于禁止转账和禁止销毁调用者持有的代币。只有具有operator权限的地址才能调用这些函数进行铸币和代币销毁（无需授权，直接调用内部函数_burn销毁指定地址的代币）操作。BRR代币铸币无上限，初始铸币数量为0。

```
29  function mint(address recipient_, uint256 amount_)
30      public
31      onlyOperator
32      returns (bool)
33  {
34      uint256 balanceBefore = balanceOf(recipient_);
35      _mint(recipient_, amount_);
36      uint256 balanceAfter = balanceOf(recipient_);
37
38      return balanceAfter > balanceBefore;
39  }
```

图 8 mint函数源码

```
41  function burn(uint256 amount) public override {
42      revert("can not burn");
43  }
44
45  function transfer(address recipient, uint256 amount) public override returns (bool){
46      revert("can not transfer");
47  }
48
49  function burnFrom(address account, uint256 amount)
50      public
51      override
52      onlyOperator
53  {
54      _burn(account, amount);
55  }
```

图 9 burn、transfer和burnFrom函数源码

- **相关函数：**mint、burn、transfer、burnFrom、onlyOperator
- **修改建议：**BoardroomRank合约中禁止了transfer函数进行转账，但未限制transferFrom函数，代币持有人仍可以通过transferFrom函数进行代币转账。确认是否需要禁止使用transferFrom函数进行转账。
- **修复结果：**项目方已忽略。
- **审计结果：**通过

5. ReferRank合约

(1) 代币基本信息

代币名称	ReferRank
代币简称	RFR
代币精度	18
代币总量	初始供应量为0（可铸币，无铸币上限，可销毁）
代币类型	HRC-20

表 5 代币基本信息

(2) 铸币/销毁/转账函数

- **业务描述：**如下图所示，RFR代币合约实现了mint和burnFrom函数用于铸币和代币销毁，以及transfer和burn函数用于禁止转账和禁止销毁调用者持有的代币。只有具有operator权限的地址

才能调用这些函数进行铸币和代币销毁（无需授权，直接调用内部函数_burn销毁指定地址的代币）操作。RFR代币铸币无上限，初始铸币数量为0。

```
29  function mint(address recipient_, uint256 amount_)
30      public
31      onlyOperator
32      returns (bool)
33  {
34      uint256 balanceBefore = balanceOf(recipient_);
35      _mint(recipient_, amount_);
36      uint256 balanceAfter = balanceOf(recipient_);
37
38      return balanceAfter > balanceBefore;
39  }
```

图 10 mint函数源码

```
41  function burn(uint256 amount) public override {
42      revert("can not burn");
43  }
44
45  function transfer(address recipient, uint256 amount) public override returns (bool){
46      revert("can not transfer");
47  }
48
49  function burnFrom(address account, uint256 amount)
50      public
51      override
52      onlyOperator
53  {
54      _burn(account, amount);
55  }
```

图 11 burn、transfer和burnFrom函数源码

- **相关函数：**mint、burn、transfer、burnFrom、onlyOperator
- **修改建议：**ReferRank合约中禁止了transfer函数进行转账，但未限制transferFrom函数，代币持有人仍可以通过transferFrom函数进行代币转账。确认是否需要禁止使用transferFrom函数进行转账。
- **修复结果：**项目方已忽略。
- **审计结果：**通过

6. STCUSDTLPTokenSharePool合约

(1) 基本信息

- **业务描述：**合约实现了stake、earned、withdraw、exit、getReward等函数用于用户去抵押指定LP代币并获取Share代币奖励以及notifyRewardAmount函数用于指定的奖励分配管理员地址rewardDistribution进行奖励相关系数的修改。奖励分为多个周期，每个周期（30天）结束后，奖励比率会更新为当前奖励比率的75%。注：合约初始化后必须调用notifyRewardAmount进行奖励比率的设置，否则会导致奖励比率异常。即使奖励比例rewardRate在checkhalve修饰器执行逻辑时被减产更新后仍可通过在该函数输入指定数值reward来修改，如数值过小会使得用户收益与预期不符。
- **相关函数：** stake、earned、withdraw、exit、getReward、notifyRewardAmount、updateReward、lastTimeRewardApplicable、rewardPerToken、totalSupply、balanceOf、safeTransfer、checkStart、checkhalve
- **审计结果：**通过

(2) 触发事件参数错误

- **业务描述：**如下图所示，STCUSDTLPTokenSharePool合约实现了notifyRewardAmount函数，用于operator初始化或重新设置合约奖励相关参数。

```
189  function notifyRewardAmount(uint256 reward)
190      external
191      override
192      onlyRewardDistribution
193      updateReward(address(0))
194  {
195      if (block.timestamp > starttime) {
196          if (block.timestamp >= periodFinish) {
197              rewardRate = reward.div(DURATION);
198          } else {
199              uint256 remaining = periodFinish.sub(block.timestamp);
200              uint256 leftover = remaining.mul(rewardRate);
201              rewardRate = reward.add(leftover).div(DURATION);
202          }
203          lastUpdateTime = block.timestamp;
204          periodFinish = block.timestamp.add(DURATION);
205          emit RewardAdded(reward);
206      } else {
207          rewardRate = initreward.div(DURATION);
208          lastUpdateTime = starttime;
209          periodFinish = starttime.add(DURATION);
210          emit RewardAdded(reward);
211      }
212  }
```

图 12 notifyRewardAmount函数源码

- **相关函数：**notifyRewardAmount、updateReward、onlyRewardDistribution
- **修改意见：**STCUSDTLPTokenSharePool合约notifyRewardAmount函数中在starttime之前调用时，未将参数对应的代币计入奖励，但是最终的RewardAdded事件又表示reward已经增加到奖励计算中。确认starttime之前的新增的reward是否需要纳入奖励，以及对应事件触发。如果不纳入奖励，则触发事件的参数应为initreward，否则应为initreward+reward。
- **修复结果：**项目方已忽略。
- **审计结果：**通过

7. STSUSDTLPTokenSharePool合约

(1) 基本信息

- **业务描述：**合约实现了stake、earned、withdraw、exit、getReward等函数用于用户去抵押指定LP代币并获取Share代币奖励以及notifyRewardAmount函数用于指定的奖励分配管理员地址rewardDistribution进行奖励相关系数的修改。奖励分为多个周期，每个周期持续365天。注：合约初始化后必须调用notifyRewardAmount进行奖励比率的设置，否则会导致奖励比率异常。指定的奖励分配管理员地址rewardDistribution可以在任意时刻调用notifyRewardAmount函数输入指定数值reward来修改，如数值过小会使得用户收益与预期不符。
- **相关函数：**stake、earned、withdraw、exit、getReward、notifyRewardAmount、updateReward、lastTimeRewardApplicable、rewardPerToken、totalSupply、balanceOf、safeTransfer、checkStart
- **审计结果：**通过

8. Operator合约

(1) 基本信息

- **业务描述：**合约实现了operator、isOperator、transferOperator等函数用于合约拥有者去进行operator权限管理。
- **相关函数：**operator、onlyOperator、isOperator、transferOperator
- **审计结果：**通过

(2) 触发事件参数错误

- **业务描述：**如下图所示，Operator合约实现了transferOperator函数，用于owner转移operator权限，函数执行成功会触发OperatorTransferred事件。

```
35  function transferOperator(address newOperator_) public onlyOwner {
36      _transferOperator(newOperator_);
37  }
38
39  function _transferOperator(address newOperator_) internal {
40      require(
41          newOperator_ != address(0),
42          'operator: zero address given for new operator'
43      );
44      emit OperatorTransferred(address(0), newOperator_);
45      _operator = newOperator_;
46  }
47  }
```

图 13 transferOperator和_transferOperator函数源码

- **相关函数：**transferOperator
- **修改意见：**函数中OperatorTransferred事件的第一个参数是固定为零地址，实际应为当前operator。建议将第44行代码中第一个参数address(0)修改为_operator。
- **修复结果：**项目方已忽略。
- **审计结果：**通过

9. Treasury合约

(1) 基本信息

- **业务描述：**合约实现了buyBonds、redeemBonds等函数用于用户去使用Cash代币购买Bond代币（要求Cash代币的价格小于变量cashPriceOne的值，当前变量cashPriceOne的值为10^{18}）或使用Bond代币赎回Cash代币（要求Cash代币的价格小于变量cashPriceCeiling的值，当前变量cashPriceOne的值为1.05×10^{18}）以及allocateSeigniorage函数用于任意用户去进行铸币税的分配（要求满足Cash代币的价格小于变量cashPriceCeiling的值)。
- **相关函数：**buyBonds、redeemBonds、balanceOf、burnFrom、safeTransfer、allocateSeigniorage、totalSupply、mint、notifyRewardAmount
- **审计结果：**通过

(2) 代币价格获取错误

- **业务描述：**如下图所示，Treasury合约实现了allocateSeigniorage函数，用于合约owner进行Cash代币的分配,_updateCashPrice函数用于更新Oracle合约中的代币价格，每当buyBonds、redeemBonds和allocateSeigniorage函数被调用时，都会调用_updateCashPrice函数进行代币价格的更新。

```
188    function _updateCashPrice() internal {
189        try IOracle(bondOracle).update()  {} catch {}
190        // try IOracle(seigniorageOracle).update()  {} catch {}
191    }
```

图 14 _updateCashPrice函数源码

```
249    function allocateSeigniorage()
250        external
251        onlyOneBlock
252        checkMigration
253        checkStartTime
254        checkEpoch
255        checkOperator
256    {
257        _updateCashPrice();
258        uint256 cashPrice = _getCashPrice(seigniorageOracle);
259        if (cashPrice <= cashPriceCeiling) {
260            return; // just advance epoch instead revert
261        }
262
263        // circulating supply
264        uint256 cashSupply = IERC20(cash).totalSupply().sub(
265            accumulatedSeigniorage
266        );
267        uint256 percentage = cashPrice.sub(cashPriceOne);
268        uint256 seigniorage = cashSupply.mul(percentage).div(1e18);
269        IStableAsset(cash).mint(address(this), seigniorage);
```

图 15 allocateSeigniorage函数部分源码

- **相关函数：**allocateSeigniorage、consult、totalSupply、mint、notifyRewardAmount、safeTransfer
- **修改意见：**由于_updateCashPrice函数中seigniorageOracle地址相关代码被注释掉了，调用allocateSeigniorage时_getCashPrice函数可能无法获取最新的Cash代币价格，建议确认相关逻辑。
- **修复结果：**项目方已忽略，因为地址seigniorageOracle和地址bondOracle属于同一个地址。
- **审计结果：**通过

10. InitialShareDistributor合约

(1) 基本信息

- **业务描述：**合约实现了distribute函数用于同时更新多个抵押池合约中的奖励相关参数。注：任意用户均可调用distribute函数，此函数只能调用一次。

- **相关函数：**distribute、transfer、notifyRewardAmount
- **审计结果：**通过

11. Boardroom合约

(1) 基本信息

- **业务描述：**合约实现了stake、earned、withdraw、exit、claimReward等函数用于用户去抵押指定代币并获取Cash代币奖励以及notifyRewardAmount函数用于指定的奖励分配管理员地址rewardDistribution进行奖励相关系数的修改。奖励分为多个周期，每个周期持续7天。注：合约初始化后必须调用notifyRewardAmount进行奖励比率的设置，否则会导致奖励比率异常。指定的奖励分配管理员地址rewardDistribution可以在任意时刻调用notifyRewardAmount函数输入指定数值reward来修改，如数值过小会使得用户收益与预期不符。
- **相关函数：**stake、earned、withdraw、exit、claimReward、notifyRewardAmount、updateReward、lastTimeRewardApplicable、rewardPerToken、totalSupply、balanceOf、safeTransfer
- **审计结果：**通过

12. Distributor合约

(1) 基本信息

- **业务描述：**合约实现了distribute函数用于批量调用InitialShareDistributor合约进行抵押池合约中的奖励相关参数的更新及发送奖励代币到该抵押池。注：任意用户均可调用distribute函数。
- **相关函数：**distribute、transfer、notifyRewardAmount
- **审计结果：**通过

13. MiningPool合约

(1) 基本信息

- **业务描述：**合约实现了stake、earned、withdraw、exit、getReward等函数用于operator去抵押指定代币并获取指定代币奖励以及notifyRewardAmount函数用于指定的奖励分配管理员地址rewardDistribution进行奖励相关系数的修改。奖励分为多个周期，每个周期持续7天。注：合约初始化后必须调用notifyRewardAmount进行奖励比率的设置，否则会导致奖励比率异常。指定的奖励分配管理员地址rewardDistribution可以在任意时刻调用notifyRewardAmount函数输入指定数值reward来修改，如数值过小会使得用户收益与预期不符。
- **相关函数：**stake、earned、withdraw、exit、getReward、notifyRewardAmount、updateReward、lastTimeRewardApplicable、rewardPerToken、totalSupply、balanceOf、safeTransfer、checkStart

- **审计结果：**通过

14. Oracle合约

(1) 基本信息

- **业务描述：**合约实现了update、consult和consultNow等函数用于获取流动性交易池中指定交易对中的代币价格。
- **相关函数：**update、consult、consultNow、currentCumulativePrices
- **审计结果：**通过

15. SimpleERCFund合约

(1) 基本信息

- **业务描述：**合约实现了deposit函数用于任意用户存入任意代币以及withdraw函数用于operator提取本合约中的任意代币。
- **相关函数：**deposit、withdraw、safeTransfer
- **审计结果：**通过

16. Timelock合约

(1) 基本信息

- **业务描述：**合约实现了setDelay、acceptAdmin、setPendingAdmin、queueTransaction、cancelTransaction、executeTransaction等函数用于合约管理者更新指定合约中的相关参数时能够延迟生效。
- **相关函数：**setDelay、acceptAdmin、setPendingAdmin、queueTransaction、cancelTransaction、executeTransaction
- **审计结果：**通过

17. VoteProxy合约

(1) 基本信息

- **业务描述：**合约实现了setBoardroom、decimals、name、symbol、totalSupply和balanceOf函数用于查询Boardroom合约中的相关信息。
- **相关函数：**setBoardroom、decimals、name、symbol、totalSupply、balanceOf
- **审计结果：**通过

结论

成都链安对 StableCash 项目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，项目方均已忽略,**StableCash 项目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