成都链安
BEOSIN

智能合约安全审计报告

成都链安 BEOSIN

审计报告编号：202012291532

报告查询名称：EDC

审计项目名称：

EarnDefiCoin (EDC)

审计合约信息：

初始审计文件哈希(SHA256)	c7db4b8860de0b58743a0d7959a89e43932d6a0afaf01c5e0d7bf07c71bcdf17
最终审计文件哈希(SHA256)	8b5614ca351d97654a2c91a08265e6bdf4405331b6e4e7187b5d6dae30a8a50a

合约审计开始日期：2020.12.22

合约审计完成日期：2020.12.29

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
		SafeMath 功能审计	通过
		require/assert 使用审计	通过
		gas 消耗审计	通过
		可见性规范审计	通过
		falback 函数使用审计	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。

免责声明：本次审计仅针对本报告载明的审计类型及结果表中给定的审计类型范围进行审计，其他未知安全漏洞不在本次审计责任范围之内。成都链安科技仅根据本报告出具前已经存在或发生的攻击或漏洞出具本报告，对于出具以后存在或发生的新的攻击或漏洞，成都链安科技无法判断其对智能合约安全状况可能的影响，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，仅基于合约提供者在本报告出具前已向成都链安科技提供的文件和资料，且该部分文件和资料不存在任何缺失、被篡改、删减或隐瞒的前提下作出的；如提供的文件和资料存在信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符等情况或提供文件和资料在本报告出具后发生任何变动的，成都链安科技对由此而导致的损失和不利影响不承担任何责任。成都链安科技出具的本审计报告系根据合约提供者提供的文件和资料依靠成都链安科技现掌握的技术而作出的，由于任何机构均存在技术的局限性，成都链安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 EDC 智能合约的代码规范性、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，EDC 智能合约通过所有检测项，合约审计结果为通过。**以下为本项目的合约详细审计信息：

1 代码规范审计

1.1 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。

本合约的编译器版本声明的相关代码被注释，导致合约编译时会出现如下图所示的编译器警告。

```
browser/tests/edc.sol: Warning: Source file does
not specify required compiler version! Consider
adding "pragma solidity ^0.7.6;"
```

图 1 EDC 合约编译器警告

- **安全建议：**消除编译器警告，指定编译器版本
- **修复结果：已修复。**
- **审计结果：通过**

1.2 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**

1.3 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。

合约中实现了Address地址库且在主合约中引用，但合约中实际上并未使用到Address库内的任何函数。属于冗余代码。

```
263 library Address {···
396 }
397
398 /**
399  * @dev Contract module which provides a basic access control mechanism, where
400  * there is an account (an owner) that can be granted exclusive access to
401  * specific functions.
402  *
403  * By default, the owner account will be the one that deploys the contract. This
404  * can later be changed with {transferOwnership}.
405  *
406  * This module is used through inheritance. It will make available the modifier
407  * `onlyOwner`, which can be applied to your functions to restrict their use to
408  * the owner.
409  */
410 contract Ownable is Context {···
460 }
461
462
463 contract Drip is Context, IERC20, Ownable {
464     using SafeMath for uint256;
465     using Address for address;
```

图 2 Address库及其引用声明

- **安全建议：**删除Address库及其引用代码。
- **修复结果：已修复。**
- **审计结果：通过**

1.4 SafeMath功能审计

检查合约中是否正确使用SafeMath库内的函数进行数学运算，或者进行其他防溢出的检查。

- **安全建议：无**
- **审计结果：通过**

1.5 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**

1.6 gas消耗审计

以太坊的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**

1.7 可见性规范审计

检查合约函数的可见性是否符合设计要求。

- **安全建议：无**
- **审计结果：通过**

1.8 fallback函数使用审计

检查在当前合约中是否正确使用了fallback函数。

- **安全建议：无**
- **审计结果：通过**

2 通用漏洞审计

2.1 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。

本合约绝大部分数学运算都使用了防溢出SafeMath库内函数，其中*_ceil*函数中并未使用防溢出安全函数进行数学运算，但分析其实际使用环境下的数值大小，不存在整型溢出风险。

```
582     function _getTValues(uint256 tAmount) private view returns (uint256, uint256) {
583         uint256 tFee = 0;
584         if (tAmount > 1) {
585             uint256 roundValue = _ceil(tAmount, _inverseFeeRate);
586             tFee = roundValue.mul(100).div(_inverseFeeRateTimes100);
587         }
588         uint256 tTransferAmount = tAmount.sub(tFee);
589         return (tTransferAmount, tFee);
590     }
```

图 3 _getTValues函数源码

- **安全建议：无**
- **审计结果：通过**

2.2 重入攻击审计

重入漏洞是最典型的智能合约漏洞，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**

2.3 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**

2.4 交易顺序依赖审计

在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**

2.5 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。

- **安全建议：无**
- **审计结果：通过**

2.6 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**

2.7 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**

2.8 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TRC20 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**

2.9 tx.origin使用安全审计

在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**

2.10 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。**

2.11 变量覆盖审计

智能合约中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**

3 业务审计

3.1 合约代币基本信息

- **业务描述：**

本合约实现了符合 ERC20 Token 标准的合约代币，其基本信息如下：

Token name	EarnDefiCoin
Token symbol	EDC
decimals	9
totalSupply	1000万枚（可销毁）
Token type	ERC20

表 1 代币基本信息

- **审计结果：通过**

3.2 合约代币基本功能

- **业务描述：**

合约代币 EDC 实现了 ERC20 Token 中要求的所有接口，包含了代币转账、代币授权、代理转账等基本功能。

本合约的 *balanceOf* 函数会根据对应地址是否被排除分红返回不同的数值。如果对应地址已经被排除，则会返回直接该地址的_tOwned 变量记录值；否则，则调用 *tokenFromReflection* 函数根据其持有到的代币映射值计算当前兑换比例下的代币数量。

```
394    function balanceOf(address account) public view override returns (uint256) {
395        if (_isExcluded[account]) return _tOwned[account];
396        return tokenFromReflection(_rOwned[account]);
397    }
```

图 4 balanceOf 函数源码

- **相关函数：** *transfer*、*transferFrom*、*approve*、*allowance*、*balanceOf*、*increaseAllowance*、*decreaseAllowance*、*tokenFromReflection*
- **安全建议：无**
- **审计结果：通过**

3.3 合约管理权限控制

- **业务描述：**

具备合约管理权限的账户(默认为合约部署者)可调用 *transferOwnership* 函数将本合约的管理权限转移至指定的非零地址；也可调用 *renounceOwnership* 函数放弃本合约的管理权限；合约当前管理员 owner 可调用合约的相关函数设置系统参数、空投和进行_excluded 列表管理等。

- **相关函数：** *transferOwnership*、*renounceOwnership*
- **安全建议：无**
- **审计结果：通过**

3.4 控制排除分红的地址列表

- **业务描述：**

合约管理员 owner 可调用合约的 *excludeAccount* 函数将指定地址添加至代币通缩分红的排除名单；也可调用 *includeAccount* 函数将排除名单中的指定地址移除。

excludeAccount 函数会将指定地址添加至代币通缩分红的排除名单，并以其当前持有的代币映射值计算出对应的代币数量。

```
621     function excludeAccount(address account) external onlyOwner() {
622         require(!_isExcluded[account], "Account is already excluded");
623         if(_rOwned[account] > 0) {
624             _tOwned[account] = tokenFromReflection(_rOwned[account]);
625         }
626         _isExcluded[account] = true;
627         _excluded.push(account);
628     }
```

图 5 excludeAccount函数源码

includeAccount 函数会将指定地址从代币通缩分红的排除名单中移除，并将其代币持有数量_tOwned 置为 0。

```
630     function includeAccount(address account) external onlyOwner() {
631         require(_isExcluded[account], "Account is already excluded");
632         for (uint256 i = 0; i < _excluded.length; i++) {
633             if (_excluded[i] == account) {
634                 _excluded[i] = _excluded[_excluded.length - 1];
635                 _tOwned[account] = 0;
636                 _isExcluded[account] = false;
637                 _excluded.pop();
638                 break;
639             }
640         }
641     }
```

图 6 includeAccount函数源码

需要注意，合约中存在*_transferNoFee* 和*_transferNormal* 函数用于处理代币转账，如果对应的地址已被排除，同样也会当前系统的兑换比例计算对应地址的代币映射值_rOwned，并进行更新。那么可能会出现这种情况：用户在较高兑换比例时被加入排除名单，待兑换比例降低后(兑换比例会逐渐减低)，将 EDC 代币全额转移至其他地址，由于兑换比例差，该用户转账时计算出的代币映射值

_rOwned 小于其被添加至排除名单时的值，那么全额转账后，该用户的代币映射值仍会有剩余，因此，如果此时该用户被从排除名单中移除，那么该用户又存在可用的代币，也就是说，该用户尽管被排除，但实际却享受到了代币通缩分红。

- **相关函数：** *excludeAccount*、*includeAccount*
- **安全建议：** 在调用*includeAccount*函数时，获取对应地址的_tOwned值和当前兑换比例，然后计算出对应_rOwned值并赋值给该地址，并更新_rTotal。
- **修复结果：已修复。**
- **审计结果：通过**

3.5 转账手续费地址管控

- **业务描述：**

合约当前管理员 owner 可调用 *setNoFeeOnSend*、*setHasFeeOnSend*、*setNoFeeOnReceive* 和 *setHasFeeOnReceive* 函数设置指定地址在发送代币和接收代币时是否需要支付手续费。

- **相关函数：** *setNoFeeOnSend*、*setHasFeeOnSend*、*setNoFeeOnReceive*、*setHasFeeOnReceive*
- **安全建议：无**
- **审计结果：通过。**

3.6 设置系统参数

- **业务描述：**

合约当前管理员 owner 可调用 *setInverseFeeRate* 函数设置转账手续费收取比例；调用 *setBurnDiv* 函数设置手续费销毁比例。

- **相关函数：** *setInverseFeeRate*、*setBurnDiv*
- **安全建议：无**
- **审计结果：通过**

3.7 减少持有代币映射值

- **业务描述：**

如下图所示，持有 EDC 代币且未被排除的用户可调用合约的 *reflect* 函数销毁自身指定数量的代币。在此函数中，以代币数量为输入参数，但实际上减少的代币映射值，该操作会将被销毁的代币价值分配到剩余代币上。

```
437    function reflect(uint256 tAmount) public {
438        address sender = _msgSender();
439        require(!_isExcluded[sender], "Excluded addresses cannot call this function");
440        (uint256 rAmount,,,,) = _getValues(tAmount);
441        _rOwned[sender] = _rOwned[sender].sub(rAmount);
442        _rTotal = _rTotal.sub(rAmount);
443        _tFeeTotal = _tFeeTotal.add(tAmount);
444    }
```

图 7 reflect函数源码

- **相关函数：** *reflect*
- **安全建议：无**
- **审计结果：通过**

3.8 代币空投

- **业务描述：**

合约管理员 owner 可调用合约的 *transferAirdrop* 函数向指定地址发送代币作为空投。

```
530     function transferAirdrop(address recipient, uint256 tAmount) public onlyOwner {
531         require(!_isExcluded[msg.sender], "The owner has been excluded");
532         require(!_isExcluded[recipient], "The recipient address has been excluded");
533         uint256 currentRate =  _getRate();
534         uint256 rAmount = tAmount.mul(currentRate);
535         _rOwned[msg.sender] = _rOwned[msg.sender].sub(rAmount);
536         _rOwned[recipient] = _rOwned[recipient].add(rAmount);
537         emit Transfer(msg.sender, recipient, tAmount);
538     }
```

图 8 transferAirdrop 函数源码

- **相关函数：** *transferAirdrop*
- **安全建议：无**
- **审计结果：通过**

3.9 关键内部函数

3.9.1 代币转账内部函数

- **业务描述：**

合约重写了*_transfer* 函数，如下图，函数增加了检查发送者和接收者是否需要收取手续费的逻辑，并进入不同的处理逻辑。

```
518     function _transfer(address sender, address recipient, uint256 amount) private {
519         require(sender != address(0), "ERC20: transfer from the zero address");
520         require(recipient != address(0), "ERC20: transfer to the zero address");
521         require(amount > 0, "Transfer amount must be greater than zero");
522         // whitelist
523         if (_noFeeOnSend[sender] || _noFeeOnReceive[recipient]) {
524             _transferNoFee(sender, recipient, amount);
525         } else {
526             _transferNormal(sender, recipient, amount);
527         }
528     }
```

图 9 _transfer函数源码

_transferNoFee 函数为不需要收取手续费的转账处理逻辑，该函数会检查发送者和接收者是否被排除份分红，如果被排除，则仅更新对应地址的_rOwned，否则还需要更新对应地址的_tOwned。

```
540     function _transferNoFee(address sender, address recipient, uint256 tAmount) private {
541         // Calculate like transferStandard, but where tFee = 0
542         uint256 currentRate =  _getRate();
543         uint256 rAmount = tAmount.mul(currentRate);
544         if (_isExcluded[sender]) {
545             _tOwned[sender] = _tOwned[sender].sub(tAmount);
546         }
547         _rOwned[sender] = _rOwned[sender].sub(rAmount);
548         if (_isExcluded[recipient]) {
549             _tOwned[recipient] = _tOwned[recipient].add(tAmount);
550         }
551         _rOwned[recipient] = _rOwned[recipient].add(rAmount);
552         emit Transfer(sender, recipient, tAmount);
553     }
```

图 10 _transferNoFee函数源码

_transferNormal 函数为需要收取手续费的转账处理逻辑，该函数前半段与_transferNoFee 函数一致，后半段增加了处理手续费的逻辑*_reflectFee* 函数。

```
555     function _transferNormal(address sender, address recipient, uint256 tAmount) private {
556         (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee) = _getValues(tAmount);
557         if (_isExcluded[sender]) {
558             _tOwned[sender] = _tOwned[sender].sub(tAmount);
559         }
560         _rOwned[sender] = _rOwned[sender].sub(rAmount);
561         if (_isExcluded[recipient]) {
562             _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
563         }
564         _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
565         _reflectFee(rFee, tFee);
566         emit Transfer(sender, recipient, tTransferAmount);
567     }
```

图 11 _transferNormal函数源码

_reflectFee 函数会分别更新_rTotal、_tTotal，以达到代币销毁的效果。其中_tTotal 的减少量为“tFee.div(_burndiv)”，未被销毁部分的代币价值将分配给剩余代币。

```
569     function _reflectFee(uint256 rFee, uint256 tFee) private {
570         _rTotal = _rTotal.sub(rFee);
571         _tFeeTotal = _tFeeTotal.add(tFee);
572         _tTotal = _tTotal.sub(tFee.div(_burndiv));
573     }
```

图 12 _reflectFee函数源码

- **相关函数：***_transfer*、*_transferNoFee*、*_transferNormal*、*_reflectFee*
- **安全建议：无**
- **审计结果：通过。**

3.9.2 计算指定代币数量对应的相关参数

- **业务描述：**

合约中的*_getValues* 函数用于获取指定代币数量所对应的代币映射值、可转账代币映射值、转账费用映射值、可转账的代币数量、转账费用。

```
575     function _getValues(uint256 tAmount) private view returns (uint256, uint256, uint256, uint256, uint256) {
576         (uint256 tTransferAmount, uint256 tFee) = _getTValues(tAmount);
577         uint256 currentRate =  _getRate();
578         (uint256 rAmount, uint256 rTransferAmount, uint256 rFee) = _getRValues(tAmount, tFee, currentRate);
579         return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee);
580     }
```

图 13 _getValues函数源码

_getTValues 函数用于获取指定代币数量所对应的可转账代币数量和转账费用。

```
582     function _getTValues(uint256 tAmount) private view returns (uint256, uint256) {
583         uint256 tFee = 0;
584         if (tAmount > 1) {
585             uint256 roundValue = _ceil(tAmount, _inverseFeeRate);
586             tFee = roundValue.mul(100).div(_inverseFeeRateTimes100);
587         }
588         uint256 tTransferAmount = tAmount.sub(tFee);
589         return (tTransferAmount, tFee);
590     }
```

图 14 _getTValues函数源码

_getRValues 函数可根据代币数量、费用和当前的兑换比例计算出对应的代币映射值、可转账的代币映射值以及转账费用映射值。

```
592     function _getRValues(uint256 tAmount, uint256 tFee, uint256 currentRate) private pure returns (uint256, uint256, uint256) {
593         uint256 rAmount = tAmount.mul(currentRate);
594         uint256 rFee = tFee.mul(currentRate);
595         uint256 rTransferAmount = rAmount.sub(rFee);
596         return (rAmount, rTransferAmount, rFee);
597     }
```

图 15 _getRValues函数源码

- **相关函数：***_getValues*、*_getTValues*、*_getRValues*
- **安全建议：无**
- **审计结果：通过。**

3.9.3 计算当前兑换比例

- **业务描述：**

合约中*_getRate* 函数用于获取合约当前代币映射值与代币数量的兑换比例，如下图所示，该函数的本质是计算代币映射总量与代币总量的比例。

```
728     function _getRate() private view returns(uint256) {
729         (uint256 rSupply, uint256 tSupply) = _getCurrentSupply();
730         return rSupply.div(tSupply);
731     }
```

图 16 _getRate函数源码

_getCurrentSupply 函数会遍历_excluded 列表中的地址，并将其持有的代币数量和映射值从当前总量中扣除。

```
604     function _getCurrentSupply() private view returns(uint256, uint256) {
605         uint256 rSupply = _rTotal;
606         uint256 tSupply = _tTotal;
607         for (uint256 i = 0; i < _excluded.length; i++) {
608             if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return (_rTotal, _tTotal);
609             rSupply = rSupply.sub(_rOwned[_excluded[i]]);
610             tSupply = tSupply.sub(_tOwned[_excluded[i]]);
611         }
612         if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
613         return (rSupply, tSupply);
614     }
```

图 17 _getCurrentSupply函数源码

- **相关函数：***_getRate*、*_getCurrentSupply*
- **安全建议：无**
- **审计结果：通过。**

4 结论

Beosin(成都链安)对 EDC 智能合约的设计和代码实现进行了详细的审计。审计团队在审计过程中发现的问题均已告知项目方修复，EDC 智能合约的总体审计结果是**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