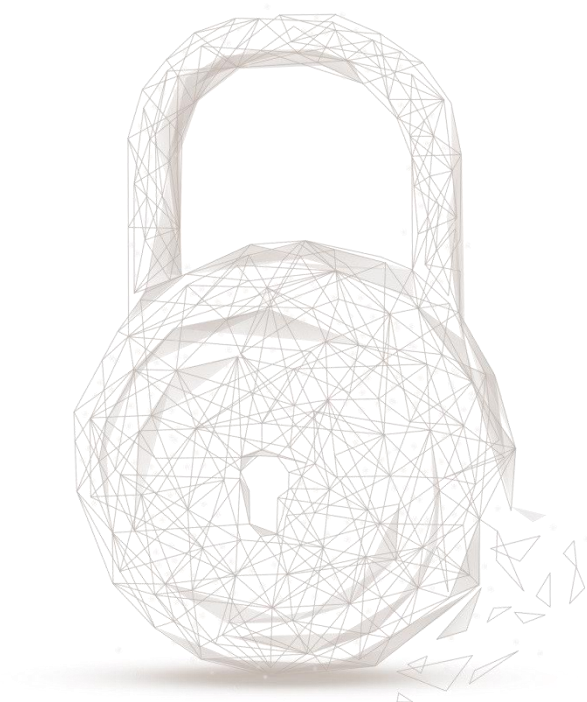




# 智能合约安全审计报告



审计编号: 202009021623

审计合约名称:

GXC & relay

审计合约链接地址:

<https://github.com/gxchain/gxc-relay-contract>

Commit Hash:

79d90eb58f45e6bb4186113e776b3a6d50831792

合约审计开始日期: 2020.08.26

合约审计完成日期: 2020.09.02

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:

### 1. ETH业务部分

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

## 2. GXC业务部分

序号	审计项目	审计结果
1	函数调用权限审计	通过
2	系统 API 函数使用审计	通过
3	get_trx_origin 使用安全审计	通过
4	溢出审计	通过
5	随机数安全审计	通过
6	手续费消耗审计	通过
7	冗余代码审计	通过
8	异常校验审计	通过
9	资产检查审计	通过
10	重放攻击审计	通过
11	业务逻辑审计	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GXC & relay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，GXC & relay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。

## ERC20代币审计：

### 1. 基本代币信息

Token name	暂无
Token symbol	暂无
decimals	5
totalSupply	初始为0(可铸币，无铸币上限；可销毁)
Token type	ERC20

### 2. 代币锁仓信息

无锁仓

## ETH业务审计：

### 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

#### 6. 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

#### 7. 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

#### 8. 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



## 9. 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

## 10. 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

## 11. 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

## 12. 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

## 13. 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

## 14. 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

## 15. 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

## 16. 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

## 17. 业务逻辑审计

### 1) 代币交付功能

- **业务描述：**该代币合约实现了deliver函数用于向指定地址转账一定数量的代币用于交付（发行）跨链资产至目标地址。该函数限制其仅可被指定DELIVER\_ROLE权限用户进行调用。合约所有者预先发送代币至该权限地址，然后通过该地址分发GXC资产（ETH链）。

```
34     function deliver(  
35         address to,  
36         uint256 amount,  
37         string memory from,  
38         string memory txid  
39     ) public {  
40         require(  
41             amount >= _minDeliver,  
42             "The minimum value must be greater than minDeliver"  
43         );  
44         require(hasRole(DELIVER_ROLE, _msgSender()), "Must have deliver role to deliver");  
45         for (uint256 i = 0; i < arrayLength; i++) {  
46             require(  
47                 keccak256(abi.encodePacked(txidArray[i])) !=  
48                 keccak256(abi.encodePacked(txid)),  
49                 "The txid has existed"  
50             );  
51         }  
52         uint256 id_number = id % arrayLength;  
53         txidArray[id_number] = txid;  
54         id++;  
55         transfer(to, amount);  
56         emit Deliver(to, amount, from, txid);  
57     }
```

图 1 deliver 函数源码截图

- **相关函数：**deliver、transfer
- **安全建议：**无

➤ **审计结果：**通过

## 2) 销毁提现功能

➤ **业务描述：**该代币合约实现了burn(uint256 amount, string memory to)函数，用户通过该函数销毁GXC资产，同时指定GXC平台账户to，在被中继服务监听到指定销毁操作后，在GXC链将其对应资产进行提现。

```
59     function burn(uint256 amount, string memory to) public {
60         require(
61             amount >= _minBurn,
62             "The minimum value must be greater than minBurn"
63         );
64         super.burn(amount);
65         emit Burn(msg.sender, amount, to);
66     }
67 }
```

图 2 burn 函数源码截图

➤ **相关函数：**burn(uint256 amount, string memory to)、burn(uint256 amount)

➤ **安全建议：**无

➤ **审计结果：**通过

## 3) 调整相关参数功能

➤ **业务描述：**该代币合约实现了adjustParams函数用于调整指定的交付（发行）与销毁功能相关限制参数。该函数限制其仅可被指定ADJUST\_ROLE权限用户进行调用。

```
68     function adjustParams(uint256 minDeliver , uint256 minBurn)
69     public
70     {
71         require(hasRole(ADJUST_ROLE, _msgSender()), "Adjust role required");
72         _minDeliver = minDeliver;
73         _minBurn = minBurn;
74     }
```

图 3 adjustParams 函数源码截图

➤ **相关函数：**adjustParams

➤ **安全建议：**无

➤ **审计结果：**通过

## GXC业务审计：

### 1. 函数调用权限审计

因项目的业务需求，智能合约一般存在着高权限要求的功能函数，如本合约的withdraw、confirmd、confirmw函数均要求调用者必须为本合约的管理员。因此，需要对合约函数的调用权限加以控制，避免权限泄露导致的安全问题。



```
44 //abi action
45 void withdraw(std::string to_account, contract_asset amount, std::string from_target, std::string txid, std::string from_account)
46 {
47     uint64_t account_id = get_account_id(to_account.c_str(), to_account.size());
48     uint64_t sender = get_trx_sender();
49     auto coin_kind = find(TARGETS.begin(), TARGETS.end(), from_target);
50     graphene_assert(amount.asset_id == 1, "Only support GXC");
51     graphene_assert(amount.amount >= MIN_WITHDRAW, "Must greater than min number");
52     graphene_assert(coin_kind != TARGETS.end(), "Invalid target");
53     graphene_assert(sender == adminAccount, "No authority");
54     graphene_assert(account_id >= 0, "Invalid account_name to_account");
55     graphene_assert(amount.amount > 0, "Invalid amount");
56     if (from_target == "ETH")
57     {
58         for(auto id_begin = eth_withdraw_table.begin(); id_begin != eth_withdraw_table.end(); id_begin++){
59             graphene_assert((*id_begin).txid != txid, "The txid is existed, be honest");
60         }
61     }
```

图 4 withdraw 部分函数源码截图

```
90 //abi action
91 void confirmd(uint64_t order_id, std::string target, std::string addr, contract_asset amount, std::string txid)
92 {
93     uint64_t sender = get_trx_sender();
94     graphene_assert(sender == adminAccount, "You have no authority");
95     auto idx = fund_in_table.find(order_id);
96     graphene_assert(idx != fund_in_table.end(), "There is no that order_id");
97     graphene_assert((*idx).target == target, "Unmatched chain name");
98     graphene_assert((*idx).asset_id == amount.asset_id, "Unmatched assert id");
99     graphene_assert((*idx).amount == amount.amount, "Unmatched assert amount");
100     if (target == "ETH")
101     {
102         for(auto id_begin = eth_confirm_table.begin(); id_begin != eth_confirm_table.end(); id_begin++){
103             graphene_assert((*id_begin).txid != txid, "The txid is existed, be honest");
104         }
105     }
```

图 5 confirmd 部分函数源码截图

- 安全建议：无
- 审计结果：通过

## 2. 系统API函数使用审计

GXChain本身为智能合约提供了大量的内置API，合约开发者在进行合约开发的过程中，一般会使用这些内置API完成复杂的业务逻辑，因此如果出现内置API使用不当，将会造成不可预估的安全问题。

- 安全建议：无
- 审计结果：通过

## 3. get\_trx\_origin使用安全审计

在 GXChain 的智能合约中，get\_trx\_origin 函数获取的是本次交易的原始调用者账户 ID；而 get\_trx\_sender 函数获取的则是本次交易的直接调用者 ID。如果将这二者混淆使用，可能会导致逻辑错误。

- 安全建议：无
- 审计结果：通过

## 4. 溢出审计

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。例如uint64\_t表示为一个0到(2\*\*64-1)的整数，其最大值为2\*\*64-1，但如果最大值加1，那么会溢出而得到0，同样地，0减去1会

下溢得到最大值。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

## 5. 随机数安全审计

智能合约中可能会使用到随机数，目前最为常见的做法是用区块信息作为随机因子生成，但是这样使用是不安全的，因为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。

- **安全建议：**无
- **审计结果：**通过

## 6. 手续费消耗审计

在GXChain的智能合约中，消耗存储空间时需要支付对应的手续费。因此在合约设计时，需考虑到手续费消耗，避免手续费消耗过高；同时合约在进行表数据添加和修改时，应该按业务逻辑合理指定其RAM支付者。

- **安全建议：**无
- **审计结果：**通过

## 7. 冗余代码审计

智能合约中的冗余代码会降低代码可读性，并可能在部署合约时需要消耗更多的GXC，建议消除代码冗余。

- **安全建议：**无
- **审计结果：**通过

## 8. 异常校验审计

GXChain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。内置函数`graphene_assert`用于检查条件并在条件不满足时抛出异常，并返回对应的错误提示信息。

- **安全建议：**无
- **审计结果：**通过

## 9. 资产检查审计

GXChain所使用资产由两部分构成：资产数量和资产ID，其中资产ID是该资产在GXChain链上的唯一标识。因此合约在进行资产交易检查时，既要检查其资产数量，还要检查资产ID。

- **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. 业务逻辑审计

### 1) 用户充值功能

- **相关函数：**deposit
- **功能描述：**通过调用该函数并且指定目标链和目标地址，并附带GXC资产，进行充值操作。该函数进行相关检查，对充值资产类型、充值数量进行了限制。该项目中继服务在监听到用户进行充值操作后，在目标链的外部中继合约上调用相关deliver函数进行目标链资产交付（发行）。

```
20 // @abi action
21 // @abi payable
22 void deposit(std::string target, std::string addr)
23 {
24     int64_t asset_amount = get_action_asset_amount();
25     uint64_t asset_id = get_action_asset_id();
26     graphene_assert(asset_id == 1, "Only support GXC ");
27     graphene_assert(asset_amount >= MIN_DEPOSIT, "Must greater than minnumber ");
28     contract_asset amount{asset_amount, asset_id};
29     uint64_t id_number = fund_in_table.available_primary_key();
30     auto coin_kind = find(TARGETS.begin(), TARGETS.end(), target);
31     graphene_assert(coin_kind != TARGETS.end(), "Invalid chain name");
32     uint64_t sender = get_trx_sender();
33     fund_in_table.emplace(sender, [&](auto &o) {
34         o.id = id_number;
35         o.from = sender;
36         o.asset_id = asset_id;
37         o.amount = asset_amount;
38         o.target = target;
39         o.to = addr;
40         o.state = 0;
41     });
42 }
```

图 6 deposit 函数源码截图

- **安全建议：**无
  - **审计结果：**通过
- ### 2) 充值确认功能
- **相关函数：**confirmd
  - **功能描述：**该函数用于确认用户充值与跨链交付（发行）转账交易，当用户充值成功，并且在外部中继合约执行完对应deliver调用后，将中继服务获取到的对应txid与充值记录order作为参数，验证充值订单。该函数限制其调用者（sender）仅可为指定的合约管理员（adminAccount），除此之外函数还对跨链目标、订单存储的资产类型、资产数量等进行检查。

```
90 // @abi action
91 void confirmd(uint64_t order_id, std::string target, std::string addr, contract_asset amount, std::string txid)
92 {
93     uint64_t sender = get_trx_sender();
94     graphene_assert(sender == adminAccount, "You have no authority");
95     auto idx = fund_in_table.find(order_id);
96     graphene_assert(idx != fund_in_table.end(), "There is no that order_id");
97     graphene_assert((*idx).target == target, "Unmatched chain name");
98     graphene_assert((*idx).asset_id == amount.asset_id, "Unmatched assert id");
99     graphene_assert((*idx).amount == amount.amount, "Unmatched assert amount");
100     if (target == "ETH")
101     {
102         for(auto id_begin = eth_confirm_table.begin(); id_begin != eth_confirm_table.end(); id_begin++){
103             graphene_assert((*id_begin).txid != txid, "The txid is existed, be honest");
104         }
105         auto id_number = eth_confirm_table.available_primary_key();
106         eth_confirm_table.emplace(sender, [&](auto &o) {
107             o.id = id_number;
108             o.txid = txid;
109         });
110         auto begin_iterator = eth_confirm_table.begin();
111         if (id_number - (*begin_iterator).id > TXID_LIST_LIMIT)
112         {
113             eth_confirm_table.erase(begin_iterator);
114         }
115         fund_in_table.modify(idx, sender, [&](auto &o) {
116             o.state = 1;
117         });
118         fund_in_table.erase(idx);
119     }
120 }
```

图 7 confirmd 函数源码截图

- 安全建议：无
  - 审计结果：通过
- 3) 用户提现请求功能
- 相关函数：withdraw
  - 功能描述：该函数用于发起用户提现请求操作，当中继服务监听到用户在对应目标链上的销毁操作时，调用该函数发起用户提现请求操作。该函数限制其调用者（sender）仅可为指定的合约管理员（adminAccount），除此之外，还对提现资产、提现数量等进行了检查。

```
44 //abi action
45 void withdraw(std::string to_account, contract_asset amount, std::string from_target, std::string txid, std::string from_account)
46 {
47     int64_t account_id = get_account_id(to_account.c_str(), to_account.size());
48     uint64_t sender = get_trx_sender();
49     auto coin_kind = find(TARGETS.begin(), TARGETS.end(), from_target);
50     graphene_assert(amount.asset_id == 1, "Only support GXC");
51     graphene_assert(amount.amount >= MIN_WITHDRAW, "Must greater than min number");
52     graphene_assert(coin_kind != TARGETS.end(), "Invalid target");
53     graphene_assert(sender == adminAccount, "No authority");
54     graphene_assert(account_id >= 0, "Invalid account_name to_account");
55     graphene_assert(amount.amount > 0, "Invalid amount");
56     if (from_target == "ETH")
57     {
58         for(auto id_begin = eth_withdraw_table.begin(); id_begin != eth_withdraw_table.end(); id_begin++){
59             graphene_assert((*id_begin).txid != txid, "The txid is existed, be honest");
60         }
61         auto id_number = eth_withdraw_table.available_primary_key();
62         eth_withdraw_table.emplace(sender, [&](auto &o) {
63             o.id = id_number;
64             o.txid = txid;
65         });
66         auto begin_iterator = eth_withdraw_table.begin();
67         if (id_number - (*begin_iterator).id > TXID_LIST_LIMIT)
68         {
69             eth_withdraw_table.erase(begin_iterator);
70         }
71         auto contract_id = current_receiver();
72         auto contract_balance = get_balance(contract_id, amount.asset_id);
73         graphene_assert(contract_balance > amount.amount, "Balance not enough");
74         //withdraw_asset(_self, account_id, amount.asset_id, amount.amount);
75         auto id_number2 = fund_out_table.available_primary_key();
76         int64_t block_time = get_head_block_time();
77         fund_out_table.emplace(sender, [&](auto &o){
78             o.id = id_number2;
79             o.to_account = account_id;
80             o.asset_id = amount.asset_id;
81             o.amount = amount.amount;
82             o.from_target = from_target;
83             o.txid = txid;
84             o.from_account = from_account;
85             o.block_time = block_time;
86         });
87     }
88 }
```

图 8 withdraw 函数源码截图

- 安全建议：无
- 审计结果：通过
- 4) 提现确认
  - 相关函数：confirmw
  - 功能描述：该函数用于确认用户提现请求，并发送对应提现资产至对应用户。中继服务将监听提现表中的内容，定时地同意提现表中已经达成的24小时确认请求，将提现操作完成。该函数限制其调用者（sender）仅可为指定的合约管理员（adminAccount）。



```
122 //abi action
123 void confirmw()
124 {
125     uint64_t sender = get_trx_sender();
126     graphene_assert(sender == adminAccount, "You have no authority");
127     int64_t block_time_now = get_head_block_time();
128     auto idx = fund_out_table.begin();
129     auto number_index = 0;
130     graphene_assert(idx != fund_out_table.end(), "There id nothing to withdraw");
131     while((idx != fund_out_table.end()) && number_index < NUMBER_LIMIT){
132         if((*idx).block_time + TIME_GAP > block_time_now){
133             break;
134         }
135         withdraw_asset(_self, (*idx).to_account, (*idx).asset_id, (*idx).amount);
136         idx = fund_out_table.erase(idx);
137         number_index++;
138     }
```

图 9 confirmw 源码截图

- 安全建议：无
- 审计结果：通过

## ETH合约源代码审计注释：

```
// SPDX-License-Identifier: MIT
```

```
pragma solidity 0.6.2; // 成都链安 // 建议固定编译器版本
```

```
pragma experimental ABIEncoderV2; // 成都链安 // 启用用来编码和解码嵌套的数组和结构体的编译器
```

```
import "@openzeppelin/contracts/presets/ERC20PresetMinterPauser.sol";
```

```
contract GXC is ERC20PresetMinterPauser {
```

```
    // 成都链安 // 声明 bytes32 常量，存储指定权限索引
```

```
    bytes32 public constant ADJUST_ROLE = keccak256("ADJUST_ROLE");
```

```
    bytes32 public constant DELIVER_ROLE = keccak256("DELIVER_ROLE");
```

```
    string[100] private txidArray; // 成都链安 // 声明定长数组 txidArray，存储 100 个指定交易 id
```

```
    uint256 arrayLength = 100;
```

```
    uint256 private id;
```

```
    uint256 private _minDeliver = 50000;
```

```
    uint256 private _minBurn = 50000;
```

```
    uint8 private decimals_ = 5;
```

```
    // 成都链安 // 声明交付事件
```

```
    event Deliver(address indexed to, uint256 amount, string from, string txid);
```

```
// 成都链安 // 声明销毁事件
event Burn(address indexed from, uint256 amount, string to);
// 成都链安 // 构造函数，初始化代币基本信息以及设置初始 ADJUST_ROLE 权限
constructor(string memory name, string memory symbol)
    public
    ERC20PresetMinterPauser(name, symbol)
{
    super._setupDecimals(decimals_); // 成都链安 // 初始化代币精度为 5
    _setupRole(ADJUST_ROLE, _msgSender()); // 成都链安 // 调整参数权限初始为部署者地址
}

// 成都链安 // 交付函数，拥有指定权限的用户向指定地址转账一定数量的代币
function deliver(
    address to,
    uint256 amount,
    string memory from,
    string memory txid
) public {
    require(
        amount >= _minDeliver,
        "The minimum value must be greater than minDeliver"
    ); // 成都链安 // 交付数量检查，要求交付数量不小于最少交付数量
    require(hasRole(DELIVER_ROLE, _msgSender()), "Must have deliver role to deliver");
// 成都链安 // 调用权限检查，要求调用者必须拥有 DELIVER_ROLE 权限
    // 成都链安 // 遍历交易 id 数组，检查指定 id 是否存在
    for (uint256 i = 0; i < arrayLength; i++) {
        require(
            keccak256(abi.encodePacked(txidArray[i])) !=
            keccak256(abi.encodePacked(txid)),
            "The txid has existed"
        ); // 成都链安 // 要求本次交付输入的 txid 不存在与当前存储的 txidArray 数组中
    }
    uint256 id_number = id % arrayLength; // 成都链安 // 获取 txidArray 数组索引
    txidArray[id_number] = txid; // 成都链安 // 存储/更新指定索引的交易 id
    id++;
    transfer(to, amount); // 成都链安 // 调用 transfer 函数，发送指定数量的代币至目标交
    付地址
    emit Deliver(to, amount, from, txid); // 成都链安 // 触发 Deliver 事件
}

// 成都链安 // 销毁函数，调用者销毁一定数量的代币
function burn(uint256 amount, string memory to) public {
    require(
        amount >= _minBurn,
        "The minimum value must be greater than minBurn"
    ); // 成都链安 // 要求销毁数量不小于最少销毁数量
    super.burn(amount); // 成都链安 // 调用父合约 burn 函数进行代币销毁
    emit Burn(msg.sender, amount, to); // 成都链安 // 触发 Burn 事件
}
```

```
// 成都链安 // 调整最少交付数量及最小销毁数量
function adjustParams(uint256 minDeliver , uint256 minBurn)
    public
{
    require(hasRole(ADJUST_ROLE, _msgSender()), "Adjust role required"); // 成都链安
// 调用权限检查，要求调用者必须拥有 ADJUST_ROLE 权限
    _minDeliver = minDeliver;
    _minBurn = minBurn;
}
// 成都链安 // 查询最少交付数量及最小销毁数量
function getParams() public view returns (uint256 ,uint256){
    return (_minDeliver, _minBurn);
}
// 成都链安 // 查询交易 id 函数
function getTxids() public view returns (string[100] memory) {
    return txidArray;
}
}
```



成都链安  
BEOSIN

官方网址

<https://lianantech.com>

电子邮箱

[vaas@lianantech.com](mailto:vaas@lianantech.com)

微信公众号

