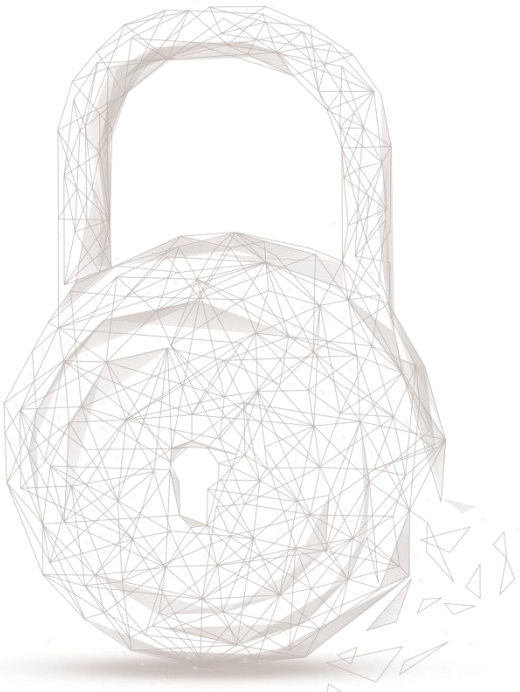
成都链安
BEOSIN

智能合约安全审计报告



成都链安 BEOSIN

审计编号：202105081129

报告查询名称：ShanHaiCard

审计合约名称：SHC

审计合约地址：0x8AB41b8AD7A5312f4Fb592B134d9b5c1e0D2dB70

审计合约链接：

https://hecoinfo.com/address/0x8AB41b8AD7A5312f4Fb592B134d9b5c1e0D2dB70#code

审计文件 hash（sha256）：
771996F2BE1FA6F301FCF7CA15875F1BC8F4ADE644683D756CCC6A532733AA69

合约审计开始日期：2021.04.26

合约审计完成日期：2021.04.29

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
		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

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Card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。**本次审计仅包含Card代币合约，经审计，Card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。

1. 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

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。

- **安全建议：无**
- **审计结果：通过**

2. 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。

- **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。

- **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。

- **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**

9. tx.origin使用安全审计

在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。

- **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

Heco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 Card代币合约审计

(1) Card代币基本信息

代币名称	ShanHaiCard
代币简称	SHC
代币总量	初始为0，可铸币，可销毁，无上限

代币类型	ERC721

表格 1 代币基本信息

(2) 授权

- **业务描述：**合约实现了 approve 和 setApprovalForAll 函数用于进行授权。approve 函数用于用户授权指定 id 代币；setApprovalForAll 函数用于用户授权自身所有资产。注意：approve 函数属性为 payable，调用的同时可发送 HT 到合约地址。调用者发送到合约里的 HT 无法提取，不建议用户发送 HT 到本合约。

```
161     function approve(address to, uint256 tokenId)
162         external payable override {
163
164         address owner = tokenOwners[tokenId];
165
166         require(msg.sender == owner
167             || approvalForAlls[owner][msg.sender],
168             "sender must be owner or approved for all"
169         );
170
171         tokenApprovals[tokenId] = to;
172         emit Approval(owner, to, tokenId);
173     }
174
175     function setApprovalForAll(address to, bool approved) external override {
176         approvalForAlls[msg.sender][to] = approved;
177         emit ApprovalForAll(msg.sender, to, approved);
178     }
```

图 1 approve 和 setApprovalForAll 函数源码

- **相关函数：**approve，setApprovalForAll
- **审计结果：通过**

(3) 转账

- **业务描述：**合约实现了 safeTransferFrom，transferFrom，safeBatchTransferFrom，batchTransferFrom 函数用于用户转账。safeTransferFrom 和 transferFrom 函数均通过调用内部函数_transferFrom 向 to 地址发送指定 id 代币，要求：1.from 和 to 均不为 0 地址；2.from 地址为代币的 owner；3.调用者具有代币操作权限。safeTransferFrom 函数和 safeBatchTransferFrom 函数会在转账完成后，判断 to 地址是否为合约地址并根据输入 data 执行相关操作。safeBatchTransferFrom 和 batchTransferFrom 函数用于向 to 地址发送多个不同 id 代币。注意：safeTransferFrom 和 transferFrom 函数属性为 payable，调用的同时可以发送 HT 到合约地址。调用者发送到合约里的 HT 无法提取，不建议用户发送 HT 到本合约。

```
84      function safeTransferFrom(address from, address to, uint256 tokenId)
85              external payable override {
86
87          safeTransferFrom(from, to, tokenId, "");
88      }
89
90      function safeTransferFrom(address from, address to, uint256 tokenId,
91          bytes memory data) public payable override {
92
93          _transferFrom(from, to, tokenId);
94
95          if (to.isContract()) {
96              require(IERC721TokenReceiver(to)
97                  .onERC721Received(msg.sender, from, tokenId, data)
98                  == Util.ERC721_RECEIVER_RETURN,
99                  "onERC721Received() return invalid");
100         }
101     }
102
103     function transferFrom(address from, address to, uint256 tokenId)
104         external payable override {
105
106         _transferFrom(from, to, tokenId);
107     }
```

图 2 safeTransferFrom 和 transferFrom 函数源码

```
109     function _transferFrom(address from, address to, uint256 tokenId)
110         internal {
111
112         require(from != address(0), "from is zero address");
113         require(to != address(0), "to is zero address");
114
115         require(from == tokenOwners[tokenId], "from must be owner");
116
117         require(msg.sender == from
118             || msg.sender == tokenApprovals[tokenId]
119             || approvalForAlls[from][msg.sender],
120             "sender must be owner or approvaled");
121
122         if (tokenApprovals[tokenId] != address(0)) {
123             delete tokenApprovals[tokenId];
124         }
125
126         _removeTokenFrom(from, tokenId);
127         _addTokenTo(to, tokenId);
128
129         emit Transfer(from, to, tokenId);
130     }
```

图 3 _transferFrom 函数源码

```
39      function safeBatchTransferFrom(address from, address to,
40          uint256[] memory tokenIds) external {
41
42          safeBatchTransferFrom(from, to, tokenIds, "");
43      }
44
45      function safeBatchTransferFrom(address from, address to,
46          uint256[] memory tokenIds, bytes memory data) public {
47
48          batchTransferFrom(from, to, tokenIds);
49
50          if (to.isContract()) {
51              require(IERC721TokenReceiverEx(to)
52                  .onERC721ExReceived(msg.sender, from, tokenIds, data)
53                  == Util.ERC721_RECEIVER_EX_RETURN,
54                  "onERC721ExReceived() return invalid");
55          }
56      }
57
58      function batchTransferFrom(address from, address to,
59          uint256[] memory tokenIds) public {
60
61          require(from != address(0), "from is zero address");
62          require(to != address(0), "to is zero address");
63
64          uint256 length = tokenIds.length;
65          address sender = msg.sender;
66
67          bool approval = from == sender || approvalForAlls[from][sender];
68
69          for (uint256 i = 0; i != length; ++i) {
70              uint256 tokenId = tokenIds[i];
71
72              require(from == tokenOwners[tokenId], "from must be owner");
73              require(approval || sender == tokenApprovals[tokenId],
74                  "sender must be owner or approvaled");
75
76              if (tokenApprovals[tokenId] != address(0)) {
77                  delete tokenApprovals[tokenId];
78              }
79
80              _removeTokenFrom(from, tokenId);
81              _addTokenTo(to, tokenId);
82
83              emit Transfer(from, to, tokenId);
84          }
85      }
```

图 4 safeBatchTransferFrom 和 batchTransferFrom 函数源码

- **相关函数：**safeTransferFrom，transferFrom，safeBatchTransferFrom，batchTransferFrom
- **审计结果：通过**

(4) 销毁

- **业务描述：**合约实现了 burn 函数用于销毁指定 id 代币，要求调用者具有其操作权限；batchBurn 函数用于销毁多个不同 id 代币，要求调用者为代币 owner；burnForSlot 函数用于销毁多个不同 id 代币，要求调用者为代币 owner，在销毁代币后，调用 Slot 合约相关函数更新数据。

```
101
102     function burn(uint256 cardId) external {
103         address owner = tokenOwners[cardId];
104
105         require(msg.sender == owner
106         || msg.sender == tokenApprovals[cardId]
107         || approvalForAlls[owner][msg.sender],
108             "msg.sender must be owner or approved");
109
110
111         _burn(cardId);
112     }
113
114     function batchBurn(uint256[] memory cardIds) external {
115         uint256 length = cardIds.length;
116         address owner = msg.sender;
117
118         for (uint256 i = 0; i != length; ++i) {
119             uint256 cardId = cardIds[i];
120             require(owner == tokenOwners[cardId], "you are not owner");
121
122             _burn(cardId);
123         }
124     }
125
126     function burnForSlot(uint256[] memory cardIds) external {
127         uint256 length = cardIds.length;
128         address owner = msg.sender;
129
130         for (uint256 i = 0; i != length; ++i) {
131             uint256 cardId = cardIds[i];
132             require(owner == tokenOwners[cardId], "you are not owner");
133             _burn(cardId);
134         }
135
136         Slot(manager.members("slot")).upgrade(owner, cardIds);
137     }
138
```

图 5 burn，batchBurn，burnForSlot 函数源码

- **相关函数：**burn，batchBurn，burnForSlot
- **审计结果：通过**

(5) 铸币

- **业务描述：**合约实现了 mint 函数用于向指定地址铸币（一次一枚）；batchMint 函数用于向指定地址一次铸多枚代币。要求调用者具有 minter 权限，将输入的 cardIdPre 经过计算得出 cardId 作为代币的 id。

```
80      function mint(address to, uint256 cardIdPre) external {
81          require(minters[msg.sender], "minter only");
82
83          uint256 cardId = NFT_SIGN_BIT | (cardIdPre & ID_PREFIX_MASK) |
84          (block.timestamp << 64) | uint64(totalSupply + 1);
85
86          _mint(to, cardId);
87      }
88
89      function batchMint(address to, uint256[] memory cardIdPres) external {
90          require(minters[msg.sender], "minter only");
91
92          uint256 length = cardIdPres.length;
93
94          for (uint256 i = 0; i != length; ++i) {
95              uint256 cardId = NFT_SIGN_BIT | (cardIdPres[i] & ID_PREFIX_MASK) |
96              (uint192(uint32(block.timestamp)) << 32) | uint32(totalSupply + 1);
97
98              _mint(to, cardId);
99          }
100     }
```

图 6 mint 和 batchMint 函数源码

- **相关函数：**mint，batchMint
- **审计结果：通过**

(6) 修改相关参数功能

- **业务描述：**合约实现了 setRarityCount 函数用于修改对应稀有度的代币的数量；setRarityFight 函数用于修改对应稀有度的代币的战力；setStarBuff 函数用于修改指定 start 的增益；setMinter 函数用于管理 minter 权限。以上函数需要调用者具有 Config 权限。

```
49      function setRarityCount(uint256 rarity, uint256 count) external CheckPermit("Config") {
50          require(rarity > 0,'min is one');
51          if(rarityCardCount.length >= rarity){
52              rarityCardCount[rarity - 1] = count;
53          }else{
54              rarityCardCount.push(count);
55          }
56      }
57
58      function setRarityFight(uint256 rarity, int256 fight) external CheckPermit("Config") {
59
60          rarityFights[rarity] = fight;
61      }
62
63      function setStarBuff(uint256 start, int256 buff) external CheckPermit("Config") {
64
65          starBuff[start] = buff;
66      }
67
68
69      function setMinter(address minter, bool enable) external CheckPermit("Config") {
70
71          minters[minter] = enable;
72      }
```

图 7 setRarityCount，setRarityFight，setStarBuff，setMinter 函数源码

- **相关函数：**setRarityCount，setRarityFight，setStarBuff，setMinter

- **审计结果：通过**

(7) 相关查询函数

- **业务描述：**合约实现了 getFight 函数用于查询指定 id 代币的战力；tokenURI 函数用于查询指定 id 代币的 URI。以及其他 ERC721 标准函数用于查询代币相关信息。

```
140     function getFight(uint256 cardId) external view returns (int256) {
141         uint256 star = (cardId ^ (1 << 191)) >> 176;
142         uint16 rarity = uint16(cardId >> 144);
143         return rarityFights[rarity].add(rarityFights[rarity].mul(starBuff[star]).div(Util.SDENO));
144     }
145 
146     function tokenURI(uint256 cardId) external view override returns (string memory) {
147 
148         return uriPrefix.concat("card/").concat(Base64.encode(Strings.toString(cardId)));
149     }
```

图 8 getFight 与 tokenURI 函数源码

```
50      function balanceOf(address owner) external view override returns(uint256) {
51          require(owner != address(0), "owner is zero address");
52          return ownerTokens[owner].length;
53      }
54 
55      // [startIndex, endIndex)
56      function tokensOf(address owner, uint256 startIndex, uint256 endIndex)
57          external view returns(uint256[] memory) {
58 
59          require(owner != address(0), "owner is zero address");
60 
61          uint256[] storage tokens = ownerTokens[owner];
62          if (endIndex == 0) {
63              return tokens;
64          }
65 
66          require(startIndex < endIndex, "invalid index");
67 
68          uint256[] memory result = new uint256[](endIndex - startIndex);
69          for (uint256 i = startIndex; i != endIndex; ++i) {
70              result[i] = tokens[i];
71          }
72 
73          return result;
74      }
75 
76      function ownerOf(uint256 tokenId)
77          external view override returns(address) {
78 
79          address owner = tokenOwners[tokenId];
80          require(owner != address(0), "nobody own the token");
81          return owner;
82      }
```

图 9 相关查询函数源码（1/2）

```
180     function getApproved(uint256 tokenId)
181         external view override returns(address) {
182
183         require(tokenOwners[tokenId] != address(0),
184             "nobody own then token");
185
186         return tokenApprovals[tokenId];
187     }
188
189     function isApprovedForAll(address owner, address operator)
190         external view override returns(bool) {
191
192         return approvalForAlls[owner][operator];
193     }
194
195     function supportsInterface(bytes4 interfaceID)
196         external pure override returns(bool) {
197
198         return interfaceID == INTERFACE_ID_ERC165
199             || interfaceID == INTERFACE_ID_ERC721
200             || interfaceID == INTERFACE_ID_ERC721Metadata;
201     }
```

图 10 相关查询函数源码（2/2）

- **相关函数：** getFight， tokenURI， balanceOf， tokensOf， ownerOf， getApproved，isApprovedForAll，supportsInterface
- **审计结果：通过**

4. 结论

Beosin(成都链安)对 Card 项目的智能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或妥善处理。Card 项目的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