FraVenner

Art Chain 智能合约
安全审计报告

FraVenner

目 录

FraVenner

1、审计概述

2021 年 7 月 24 日，FraVenner 安全团队收到 Art Chain 团队对 Art Chain 智能合约安全审计需求，根据项目实际情况，第一时间制定了审计计划并执行，最终出具安全审计报告。FraVenner 安全团队核心通过黑盒测试、白盒测试、沙盒测试等方式，模拟黑客攻击对 Art Chain 项目进行全面的安全测试与审计。

1.1 项目信息

Art Chain 是一种基于 BEP（BSC）网络发型的完全分散的社区自治令牌。代币芯片通过预售分散在每个用户手中。代币模型写在智能合约中，不可篡改，这决定了吴庄和大家庭的操纵。它是一种价值代币，100%在市场上流通，由社区共识构建。

合约名称： ART Chain

审计合约文件： 项目源代码

审计初始版本地址链接： https://github.com/bbjs123/artContracts

主要合约地址： 0xe97364f13Fc8f40B0ab372a51b645a648f091483

1.2 审计结果

审计结论： 低风险，正常通过

审计代码： 202107300028

合约审计开始时间： 2021 年 7 月 24 日

合约审计结束时间： 2021 年 7 月 30 日

审计团队： FraVenner 安全团队

FraVenner

2、审计详情

2.1 审计类型与结果

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 函数审计	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

2.2 审计结果说明

FraVenner 安全团队采用形式化验证、静态分析、动态分析、典型案例测试和人工审核的方式对 Art Chain 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，Art Chain 智能合约通过所有检测项，合约审计结果为正常通过。以下为本合约详细审计信息：

2.2.1 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.SafeMath 函数审计

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

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

6.gas 消耗审计

在虚拟机上执行合约代码需要消耗 gas，当 gas 不足时，代码执行会抛出 outofgas 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

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

8.falback 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

2.2.2 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 Heco 智能合约漏洞，曾导致了 TheDAO 被攻击。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

在进行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 DenialofService，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函数，如：铸币、自毁、changeowner 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

data 是由本合约内部构建(不可由用户任意指定)，同时检查了对应的调用返回值，无安全风险。

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 returnfalse，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 HRC20Token 的 transfer/transferFrom 函数实现中，也要避免转账失败 returnfalse 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

在相关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

虚拟机上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

2.2.3 业务逻辑审计

初次审计时，FraVenner 安全团队发现了 46 个问题，其中包含 0 个高危漏洞、39 个中危漏洞，7 个低危漏洞。经沟通，除一些可以忽略的低危风险外，项目方已整改了所有中危风险，

目前项目风险较低，正常通过。

2.2.3.1 高危漏洞

项目中不存在高危漏洞。

2.2.3.2 中危漏洞

1、函数可以标记为外部函数。

（1）“owner”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置： art.sol 文件，第 461 行

```
* @dev Returns the address of the current owner.
*/
function owner() public view returns (address) {
return _owner;
}
/**
```

修复状态： 与项目方沟通后，项目方已修复上述问题。

（2）“renounceOwnership”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置： art.sol 文件，第 48 行

```
thereby removing any functionality that is only available to the owner.
*/
function renounceOwnership() public virtual onlyOwner {
emit OwnershipTransferred(_owner, address(0));
_owner = address(0);
}
```

```
/**
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（3）“transferOwnership”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：art.sol 文件，第 489 行

```
* Can only be called by the current owner.
*/
function transferOwnership(address newOwner) public virtual onlyOwner {
require(newOwner != address(0), "Ownable: new owner is the zero address");
emit OwnershipTransferred(_owner, newOwner);
_owner = newOwner;
}
```

（4）“setUsdtAddress”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：art.sol 文件，第 582 行

```
event IntroLog(address indexed user,uint256 amount,uint256 time,address indexed intro);
function setOpenGetTime( uint256 _openGetTime) public onlyOwner{
openGetTime = _openGetTime;
}
function setUsdtAddress( address _usdtAddress) public onlyOwner{
usdtAddress = _usdtAddress;
}
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（5）“setUsdtAddress”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置： art.sol 文件，第 586 行

```
}
function setUsdtAddress( address _usdtAddress) public onlyOwner{
usdtAddress = _usdtAddress;
}
function setTokenAddress( address _tokenAddress) public onlyOwner{
tokenAddress = _tokenAddress;
```

修复状态： 与项目方沟通后，项目方已修复上述问题。

（6）“setTokenAddress”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置： art.sol 文件，第 589 行

```
usdtAddress = _usdtAddress;
}
function setTokenAddress( address _tokenAddress) public onlyOwner{
tokenAddress = _tokenAddress;
}
function setReceiveAddress( address _receiveAddress) public onlyOwner{
receiveAddress = _receiveAddress;
```

修复状态： 与项目方沟通后，项目方已修复上述问题。

（7）“setReceiveAddress”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置： art.sol 文件，第 592 行

```
}
function setReceiveAddress( address _receiveAddress) public onlyOwner{
receiveAddress = _receiveAddress;
}
function setPrice( uint256 _price) public onlyOwner{
```

```
price = _price;
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（8）“setPrice”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：art.sol 文件，第 595 行

```
receiveAddress = _receiveAddress;

}

function setPrice( uint256 _price) public onlyOwner{

price = _price;

}

function setMinnum( uint256 _minnum) public onlyOwner{

minnum = _minnum;

}
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（9）“setMinnum”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：art.sol 文件，第 598 行

```
price = _price;

}

function setMinnum( uint256 _minnum) public onlyOwner{

minnum = _minnum;

}

function setMaxnum( uint256 _maxnum) public onlyOwner{

maxnum = _maxnum;
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（10）“setMaxnum”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：art.sol 文件，第 601 行

```
minnum = _minnum;
}
function setMaxnum( uint256 _maxnum) public onlyOwner{
maxnum = _maxnum;
}
function setRecRatio( uint256 _recRatio) public onlyOwner{
recRatio = _recRatio;
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（11）“setRecRatio”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：art.sol 文件，第 604 行

```
maxnum = _maxnum;
}
function setRecRatio( uint256 _recRatio) public onlyOwner{
recRatio = _recRatio;
}
/*
* 投资
*/
function invest (uint256 amount, address intro) public nonReentrant {
require(haveBuyAmount != totalAmount,'预售已经结束');
require(amount > 0,'购买数量必须大于 0');
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（12）“invest”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子

函数都不会直接调用它。考虑把它标记为“external”。

代码位置： art.sol 文件，第 612 行

```
* 投资
*/
function invest (uint256 amount, address intro) public nonReentrant {
require(haveBuyAmount != totalAmount,'预售已经结束');
require(amount > 0,'购买数量必须大于 0');
require(amount >= minnum,'起步购买 1 份');
require(userCount[msg.sender].add(amount) <= maxnum, '不能超过最大限额');
require(haveBuyAmount.add(amount) <= totalAmount,'预售余额不足');
if(parents[msg.sender] == address(0) && intro != address(0)){
parents[msg.sender] = intro;
}
uint256 decimals = IERC20(usdtAddress).decimals();
uint usdtAmount = amount.mul(price).mul(10**decimals).div(10**18);
require(IERC20(usdtAddress).balanceOf(msg.sender) >= usdtAmount,'USDT 代币数量不足');
IERC20(usdtAddress).safeTransferFrom(msg.sender, receiveAddress, usdtAmount);
userCount[msg.sender] = userCount[msg.sender].add(amount);
//推荐认购奖励
if(parents[msg.sender] != address(0)){
introRewards[parents[msg.sender]]                                                    =
introRewards[parents[msg.sender]].add(amount.mul(recRatio).div(100));
emit IntroLog(msg.sender,amount.mul(recRatio).div(100),block.timestamp,parents[msg.sender]);
}
//已认购总量
haveBuyAmount = haveBuyAmount.add(amount);
//投资日志
emit InvestLog(msg.sender,amount,block.timestamp,intro);
```

```
}
function withdrawToken()public nonReentrant{
//require(haveBuyAmount == totalAmount,'预售尚未结束');
require(openGetTime<now,'尚未开放领取');
uint256 amount = userCount[msg.sender];
require(amount>0 , '余额不足');
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（13）“withdrawToken”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：art.sol 文件，第 642 行

```
//require(haveBuyAmount == totalAmount,'预售尚未结束');
require(openGetTime<now,'尚未开放领取');
uint256 amount = userCount[msg.sender];
require(amount>0 , '余额不足');
userCount[msg.sender] = 0;
IERC20(tokenAddress).safeTransfer(msg.sender, amount);
emit withdrawTokenLog(msg.sender,amount,block.timestamp);
}
function withdrawReward()public nonReentrant{
//require(haveBuyAmount == totalAmount,'预售尚未结束');
require(openGetTime<now,'尚未开放领取');
uint256 amount = introRewards[msg.sender];
require(amount>0 , '推荐认购奖励余额不足');
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（14）“renounceOwnership”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 433 行

```
* thereby removing any functionality that is only available to the owner.
*/
function renounceOwnership() public virtual onlyOwner {
emit OwnershipTransferred(_owner, address(0));
_owner = address(0);
}
/**
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（15）“transferOwnership”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 442 行

```
* Can only be called by the current owner.
*/
function transferOwnership(address newOwner) public virtual onlyOwner {
require(newOwner != address(0), "Ownable: new owner is the zero address");
emit OwnershipTransferred(_owner, newOwner);
_owner = newOwner;
}
function geUnlockTime() public view returns (uint256) {
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（16）“geUnlockTime”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 448 行

```
}
function geUnlockTime() public view returns (uint256) {
return _lockTime;
}
```

```
//Locks the contract for owner for the amount of time provided
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（17）“lock”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 453 行

```
//Locks the contract for owner for the amount of time provided
function lock(uint256 time) public virtual onlyOwner {
_previousOwner = _owner;
_owner = address(0);
_lockTime = now + time;
emit OwnershipTransferred(_owner, address(0));
}
//Unlocks the contract for owner when _lockTime is exceeds
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（18）“unlock”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 461 行

```
//Unlocks the contract for owner when _lockTime is exceeds
function unlock() public virtual {
require(_previousOwner == msg.sender, "You don't have permission to unlock");
require(now > _lockTime , "Contract is locked until 7 days");
emit OwnershipTransferred(_owner, _previousOwner);
_owner = _previousOwner;
}
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（19）“name”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 768 行

```
}
function name() public view returns (string memory) {
return _name;
}
function symbol() public view returns (string memory) {
return _symbol;
}
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（20）“symbol”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 772 行

```
}
function symbol() public view returns (string memory) {
return _symbol;
}
function decimals() public view returns (uint8) {
return _decimals;
}
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（21）“decimals”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 776 行

```
}

function decimals() public view returns (uint8) {

return _decimals;

}

function totalSupply() public view override returns (uint256) {

return _tTotal;

}
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（22）“totalSupply”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 780 行

```
}

function totalSupply() public view override returns (uint256) {

return _tTotal;

}

function balanceOf(address account) public view override returns (uint256) {

if (_isExcluded[account]) return _tOwned[account];

return tokenFromReflection(_rOwned[account]);
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（23）“transfer”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 789 行

```
}

function transfer(address recipient, uint256 amount) public override returns (bool) {

_transfer(_msgSender(), recipient, amount);

return true;

}
```

```
function allowance(address owner, address spender) public view override returns (uint256) {
return _allowances[owner][spender];
}
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（24）“allowance”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 794 行

```
}
function allowance(address owner, address spender) public view override returns (uint256) {
return _allowances[owner][spender];
}
function approve(address spender, uint256 amount) public override returns (bool) {
_approve(_msgSender(), spender, amount);
return true;
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（25）“approve”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 798 行

```
}
function approve(address spender, uint256 amount) public override returns (bool) {
_approve(_msgSender(), spender, amount);
return true;
}
function transferFrom(address sender, address recipient, uint256 amount) public override returns
(bool) {
_transfer(sender, recipient, amount);
_approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
```

```
amount exceeds allowance"));
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（26）“transferFrom”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 803 行

```
}

function transferFrom(address sender, address recipient, uint256 amount) public override returns
(bool) {
_transfer(sender, recipient, amount);
_approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
amount exceeds allowance"));
return true;
}
function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool)
{
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（27）“increaseAllowance”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 809 行

```
function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool)
{
_approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
return true;
}
function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns
```

```
(bool) {
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（28）“decreaseAllowance”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 814 行

```
}
function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns
(bool) {
_approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue,
"ERC20: decreased allowance below zero"));
return true;
}
function isExcludedFromReward(address account) public view returns (bool) {
return _isExcluded[account];
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（29）“isExcludedFromReward”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 819 行

```
}
function isExcludedFromReward(address account) public view returns (bool) {
return _isExcluded[account];
}
function totalFees() public view returns (uint256) {
return _tFeeTotal;
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（30）“totalFees”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 823 行

```
}
function totalFees() public view returns (uint256) {
return _tFeeTotal;
}
function deliver(uint256 tAmount) public {
address sender = _msgSender();
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（31）“deliver”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 827 行

```
}
function deliver(uint256 tAmount) public {
address sender = _msgSender();
require(!_isExcluded[sender], "Excluded addresses cannot call this function");
(uint256 rAmount,,,,,) = _getValues(tAmount);
_rOwned[sender] = _rOwned[sender].sub(rAmount);
_rTotal = _rTotal.sub(rAmount);
_tFeeTotal = _tFeeTotal.add(tAmount);
}
function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public view
returns(uint256) {
require(tAmount <= _tTotal, "Amount must be less than supply");
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（32）“reflectionFromToken”的功能定义标记为“public”。但是，同一契约中的另一个

函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 836 行

```
}
function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public view
returns(uint256) {
require(tAmount <= _tTotal, "Amount must be less than supply");
if (!deductTransferFee) {
(uint256 rAmount,,,,,) = _getValues(tAmount);
return rAmount;
} else {
(,uint256 rTransferAmount,,,,) = _getValues(tAmount);
return rTransferAmount;
}
}
function tokenFromReflection(uint256 rAmount) public view returns(uint256) {
require(rAmount <= _rTotal, "Amount must be less than total reflections");
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（33）“excludeFromReward”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 853 行

```
}
function excludeFromReward(address account) public onlyOwner() {
// require(account != 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D, 'We can not exclude Uniswap
router.');
require(!_isExcluded[account], "Account is already excluded");
if(_rOwned[account] > 0) {
_tOwned[account] = tokenFromReflection(_rOwned[account]);
```

```
}
_isExcluded[account] = true;
_excluded.push(account);
}
function includeInReward(address account) external onlyOwner() {
require(_isExcluded[account], "Account is already excluded");
for (uint256 i = 0; i < _excluded.length; i++) {
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（34）“excludeFromFee”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 886 行

```
}
function excludeFromFee(address account) public onlyOwner {
_isExcludedFromFee[account] = true;
}
function includeInFee(address account) public onlyOwner {
_isExcludedFromFee[account] = false;
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（35）“includeInFee”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 890 行

```
}
function includeInFee(address account) public onlyOwner {
_isExcludedFromFee[account] = false;
}
function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
_taxFee = taxFee;
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（36）“setSwapAndLiquifyEnabled”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 920 行

```
}

function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {

swapAndLiquifyEnabled = _enabled;

emit SwapAndLiquifyEnabledUpdated(_enabled);

}

//to recieve ETH from uniswapV2Router when swaping

receive() external payable {}
```

修复状态：与项目方沟通后，项目方已修复上述问题。

（37）“isExcludedFromFee”的功能定义标记为“public”。但是，同一契约中的另一个函数或其任何子函数都不会直接调用它。考虑把它标记为“external”。

代码位置：arttoken.sol 文件，第 1068 行

```
}

function isExcludedFromFee(address account) public view returns(bool) {

return _isExcludedFromFee[account];

}

function _approve(address owner, address spender, uint256 amount) private {

require(owner != address(0), "ERC20: approve from the zero address");
```

2、外部调用后读取持久状态。

在对用户定义的地址进行外部调用后，将访问合同帐户状态。为了防止重入问题，考虑只在调用之前访问状态，特别是如果被调用方不可信。或者，可以使用重入锁来防止不受信任的被调用方在中间状态下重新进入契约。

代码位置：arttoken.sol 文件，第 763 行

```
//exclude owner and this contract from fee
_isExcludedFromFee[owner()] = true;
_isExcludedFromFee[address(this)] = true;
emit Transfer(address(0), _msgSender(), _tTotal);
```

修复状态：与项目方沟通后，项目方已修复上述问题。

3、外部调用后写入持久状态。

在对用户定义的地址进行外部调用后，将访问合同帐户状态。为了防止重入问题，考虑只在调用之前访问状态，特别是如果被调用方不可信。或者，可以使用重入锁来防止不受信任的被调用方在中间状态下重新进入契约。

代码位置：arttoken.sol 文件，第 765 行

```
//exclude owner and this contract from fee
_isExcludedFromFee[owner()] = true;
_isExcludedFromFee[address(this)] = true;
emit Transfer(address(0), _msgSender(), _tTotal);
}
```

修复状态：与项目方沟通后，项目方已修复上述问题。

2.2.3.3 低危漏洞

1、设置了浮动杂注。

（1）当前的 pragma Solidity 指令为“^0.6.0”。建议指定一个固定的编译器版本，以确保生成的字节码在不同版本之间不存在差异。如果依赖字节码级别的代码验证，这一点尤其重要。

代码位置：art.sol 文件，第 2 行

```
// SPDX-License-Identifier: SimPL-2.0
pragma solidity ^0.6.0;
// import '@openzeppelin/contracts/access/Ownable.sol';
```

修复状态：低危风险不影响项目安全，项目方已忽略上述问题。

（2）当前的 pragma Solidity 指令为“^0.6.0”。建议指定一个固定的编译器版本，以确保生成的字节码在不同版本之间不存在差异。如果依赖字节码级别的代码验证，这一点尤其重要。

代码位置：art.sol 文件，第 497 行

```
pragma solidity ^0.6.0;


/**
```

（3）当前的 pragma Solidity 指令为“^0.6.12”。建议指定一个固定的编译器版本，以确保生成的字节码在不同版本之间不存在差异。如果依赖字节码级别的代码验证，这一点尤其重要。

代码位置：arttoken.sol 文件，第 3 行

```
pragma solidity ^0.6.12;
// SPDX-License-Identifier: Unlicensed
interface IERC20 {
```

修复状态：低危风险不影响项目安全，项目方已忽略上述问题。

2、不符合要求。

嵌套调用中不符合相关要求，因此该调用被还原。建议确保为嵌套调用提供有效的输入（例如，通过传递的参数）。

代码位置：art.sol 文件，第 335 行

```
// solhint-disable-next-line avoid-low-level-calls
(bool success, bytes memory returndata) = target.call.value(weiValue)(data);
if (success) {
```

修复状态：低危风险不影响项目安全，项目方已忽略上述问题。

3、执行对用户提供地址的调用。

执行对调用方指定地址的外部消息调用。请注意，被调用方帐户可能包含任意代码，并且可以重新输入此协定中的任何函数。以中间状态重新输入协定可能会导致意外行为。确保在此调用和/或重入保护到位后，不会执行任何状态修改。

代码位置：art.sol 文件，第 755 行

```
IUniswapV2Router02 _uniswapV2Router
= IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E);
// Create a uniswap pair for this new token
uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
.createPair(address(this), _uniswapV2Router.WETH());
// set the rest of the contract variables
uniswapV2Router = _uniswapV2Router;
```

修复状态：低危风险不影响项目安全，项目方已忽略上述问题。

4、在同一事务中执行多个调用。

此调用在同一事务中的另一个调用之后执行。如果先前的调用永久失败，则可能永远不会执行该调用。这可能是恶意被调用方故意造成的。如果可能，重构代码，使每个事务只执行一个外部调用，或者确保所有被调用方都是可信的（即，它们是您自己的代码库的一部分）。

代码位置：art.sol 文件，第 758 行

```
.createPair(address(this), _uniswapV2Router.WETH());

// set the rest of the contract variables

uniswapV2Router = _uniswapV2Router;
```

修复状态：低危风险不影响项目安全，项目方已忽略上述问题。

5、采用固定量的 Gas 费。

函数调用的是固定量的 Gas 费。不建议这样做，因为 EVM 指令的 Gas 成本在未来可能会发生变化，这可能会打破本合约的假设。如果这样做是为了防止重入攻击，则需考虑替代方法，例如检查效应交互模式或重新入选锁。

代码位置：art.sol 文件，第 1186 行

```
payable(receiveAddress).transfer(address(this).balance);

}

}

//this method is responsible for taking all fee, if takeFee is true

function _tokenTransfer(address sender, address recipient, uint256 amount,bool takeFee) private

{

if(!takeFee)
```

修复状态：低危风险不影响项目安全，项目方已忽略上述问题。

3、免责申明

FraVenner 安全团队仅就本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后发生或存在的事实，FraVenner 安全团队无法判断其智能合约安全状况，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，仅基于信息提供者截至本报告出具时向 FraVenner 安全团队提供的文件和资料（简称“已提供资料”）。FraVenner 安全团队假设：已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被

篡改、删减、隐瞒或反映的情况与实际情况不符的，FraVenner 安全团队对由此而导致的损失和不利影响不承担任何责任。

FraVenner

推特： @fra_venner

脸书： @Fra Venner

邮箱：FraVenner@outlook.com