2019

智能合约安全审计报告

MEER-DESTORY 智能合约安全审计报告

安全等级 ： ★ ★ ★ ★ ★

【文档信息】

项目	描述
文档名称	Meer-Destory 智能合约安全审计报告
基本说明	Meer-Destory 智能合约安全审计报告
修订时间	2019 年 09 月 24 日
扩散范围	Qitmeer 项目官方公开
文档编号	Qitmeer-F1DFDE70

【版权声明】

【信息反馈】

地址：深圳市深南大道佳嘉豪商务大厦 18A | 西安市丈八一路汇鑫国际 IBC-A 座 906

邮箱：support@noneage.com

电话：15029229543

前言

近年来，区块链技术受到各界的广泛关注，搜索指数持续上升，成为近年来炙手可热的新兴互联网技术之一。

安全问题一直是信息化社会的主旋律，伴随着区块链技术的不断发展，区块链领域本身的安全问题逐渐凸显，随着区块链技术在各行业领域的不断应用，以及区块链去中心化，匿名性，不可逆等一系列特点，区块链相关平台及应用相关的安全事件层出不穷。从之前的区块链底层安全技术研究曝光，发展到后来越来越多的数字货币被盗，数字货币交易平台被黑客攻击，频繁出现的智能合约漏洞，用户账户被盗等事件。

在区块链安全事件当中，智能合约漏洞导致的安全事件占所有安全事件的 31%左右，智能合约攻击导致的经济损失占所有区块链安全事件造成损失的 50%左右，智能合约安全问题涉及的问题比较复杂，涉及到区块链共识协议，智能合约语言，智能合约编译器，智能合约 gas 机制等等，软件工程师创造一个完全无误差的代码是不可能的，程序员总存在疏忽的地方，所以智能合约安全审计通常需要专业的安全团队进行深度的代码审计，成就完美合约。

为了保证用户数字货币的资产安全，提前发现并解决智能合约的安全漏洞避免导致数字货币被盗，零时科技基于团队十余年丰富的安全攻防经验，以及代码审计能力，借助自主研发的智能合约安全自动化审计系统，结合安全专家人工测试，为智能合约提供全面、深度的安全审计并提供详细审计报告及漏洞修复建议。

目录

一、综述信息

本报告根据 Qitmeer 项目官方提供的 Meer-Destroy 智能合约源代码进行安全审计，包括代码的安全漏洞挖掘和编码规范审计，并以此作为本报告的统计依据。

本次测试为非生产环境测试，所有操作均在线下测试环境进行，安全审计过程中及时跟相关接口人进行沟通，保持信息对称，在操作风险可控的情况下进行安全测试工作，以规避在测试过程中对产生和运营造成风险。

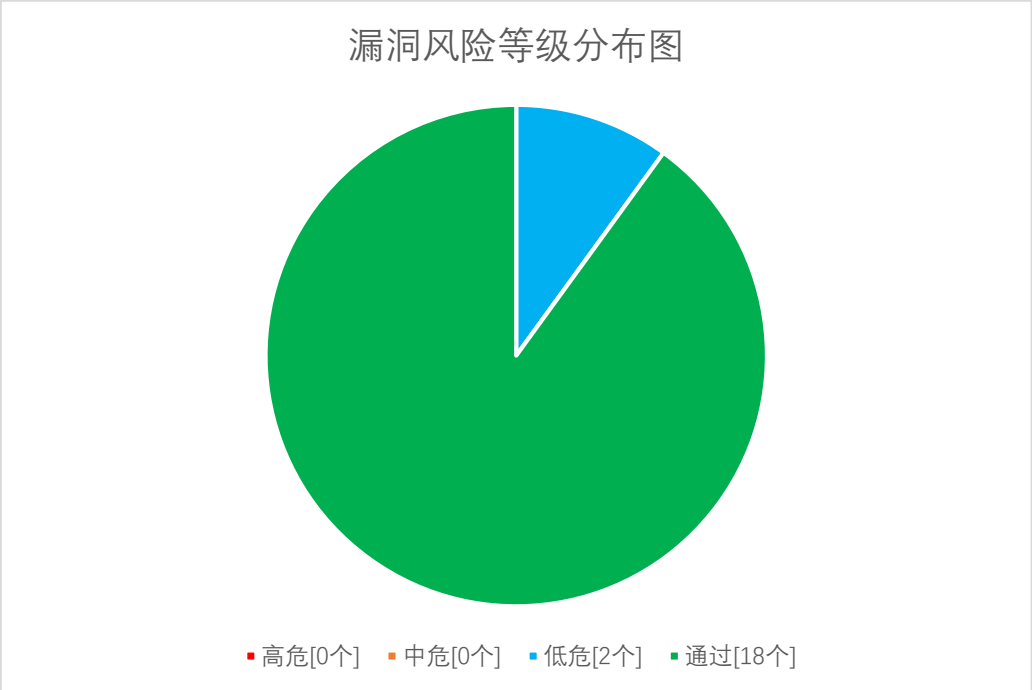
项目	描述
合约名称	Meer-Destroy
合约类型	代币合约
代码语言	Solidity
合约文件	Meer-Destroy.sol
合约地址	
审计人员	零时科技安全团队
审计时间	2019-09-24
审计工具	https://audit.noneage.com

二、代码审计结果

漏洞分布

本次安全审计漏洞风险按危险等级分布:

漏洞风险等级分布			
高危	中危	低危	通过
0	0	2	18



本次智能合约审计结果风险等级 ： ★ ★ ★ ★ ★

本次安全审计高危漏洞 0 个，中危 0 个，低危 2 个，通过 18 高，安全等级高。

审计结果

本次安全审计测试项 20 项，测试项如下：

合约名称	测试项目	项目描述	状态
	整数溢出	检查是否使用 Safemath 安全方法	**通过**
	重入攻击	检查 call.value 使用安全	**通过**
	浮点数和数值精度	检查浮点数使用安全	**通过**
	非预期的 Ether	检查是否存在强制接收 Ether	**通过**
	默认可见性	检查函数可见性设置是否正确	**通过**
	Tx.origin 身份验证	检查 tx.origin 的使用安全	**通过**
	错误的构造函数	检查合约构造函数正确性	**通过**
	未验证返回值	检查是否验证返回值	**通过**
	不安全的随机数	检查是否使用可控随机数种子	**通过**
Qitmeer 合约	**时间戳依赖**	检查逻辑判断是否依赖时间戳	**通过**
	交易顺序依赖	检查是否存在交易顺序依赖	**通过**
	Delegatecall 调用	检查 Delegatecall 函数使用安全	**通过**
	Call 调用	检查 call 函数使用安全	**通过**
	拒绝服务	检查是否存在拒绝服务的逻辑	**低危**
	逻辑设计缺陷	检查业务逻辑是否合规	**低危**
	假充值漏洞	检查是否存在假充值漏洞	**通过**
	短地址攻击	检查是否验证输入地址合法性	**通过**
	未初始化的存储指针	检查是否存在未初始化的存储指针	**通过**
	代币增发	检查是否存在代币增发接口和权限	**通过**
	冻结账户绕过	检查是否存在冻结账户绕过漏洞	**通过**

三、合约代码

代码及标注

在每一个合约代码中相应位置，都已通过注释的形式标注出安全漏洞以及编码规范问题，注释标志以**/**noneage security**/**开始，具体见合约代码内容。

合约文件：Meer-Destroy.sol

（代码见附件）

四、代码审计详情

整数溢出

漏洞描述

整数溢出一般分为又分为上溢和下溢，在智能合约中出现整数溢出的类型包括三种：乘法溢出、加法溢出、减法溢出。在 Solidity 语言中，变量支持的整数类型步长以 8 递增，支持从 uint8 到 uint256，以及 int8 到 int256，在以太坊虚拟机（EVM）中为整数指定固定大小的数据类型，而且是无符号的，例如，一个 uint8 类型 ，只能存储在范围 0 到 2^8-1，也就是[0,255] 的数字，一个 uint256 类型 ，只能存储在范围 0 到 2^256-1 的数字。这意味着在以太坊虚拟机中一个整型变量只能有一定范围的数字表示，不能超过这个制定的范围，超出变量类型所表达的数值范围将导致整数溢出漏洞。

审计结果 ： 【通过】

安全建议 ： 无

重入攻击

漏洞描述

攻击者在 Fallback 函数中的外部地址处构建一个包含恶意代码的合约，当合约向此地址发送 Ether 时，它将调用恶意代码，Solidity 中的 call.value()函数在被用来发送 Ether 时会消耗他接收到的所有 gas，所以当调用 call.value()函数发送 Ether 的操作发生在实际减少发送者账户余额之前时，将会产生重入攻击。由于重入漏洞导致了著名的 The DAO 攻击事件。

审计结果 ： 【通过】

安全建议 ： 无

浮点数和数值精度

漏洞描述

在 Solidity 中不支持浮点型，也不完全支持定长浮点型，除法运算的结果会四舍五舍，如果出现小数，小数点后的部分都会被舍弃，只取整数部分，例如直接用 5 除以 2，结果为 2。如果在代币的运算中出现运算结果小于 1Ether 的情况，比如 4.9 个代币也会被约等于 4 个，带来一定程度上的精度流失。由于代币的经济属性，精度的流失就相当于资产的流失，所以这在交易频繁的代币上会带来积少成多的问题。

审计结果 ： 【通过】

安全建议 ： 无

非预期的 Ether

漏洞描述

在 Solidity 中可以使用合约自毁函数 selfdestruct(address)功能和把 Ether 预装进合约地址这两种方法强制发送 Ether 到合约，而无需调用任何 payable 函数，如果我们的合约中错误运用

this.balance，在合约的设计逻辑中依赖于合约余额的确切值，因为它可以被人为地操纵，比如通过恶意合约代码强制支付 Ether 到合约地址从而修改 this.balance 的值，所以将导致严重漏洞。

审计结果 ： 【通过】

安全建议 ： 无

默认可见性

漏洞描述

在 Solidity 中，合约函数的可见性默认是 public。因此，不指定任何可见性的函数就可以由用户在外部调用。当开发人员错误地忽略应该是私有的功能的可见性说明符时，或者是只能在合约本身内调用的可见性说明符时，将导致严重漏洞。在 Parity 多签名钱包遭受的第一次黑客攻击中就是因为未设置函数的可见性，默认为 public，导致大量资金被盗。

审计结果 ： 【通过】

安全建议 ： 无

tx.origin 身份验证

漏洞描述

tx.origin 是 Solidity 的一个全局变量，它遍历整个调用栈并返回最初发送调用（或事务）的帐户的地址。在智能合约中使用此变量进行身份验证会使合约容易受到类似网络钓鱼的攻击。

审计结果 ： 【通过】

安全建议 ： 无

错误的构造函数

漏洞描述

在 Solidity 智能合约代码中，如果合约名称被修改，或者在构造函数名称中存在拼写错误以致构造函数的名称与合约名称不完全一致，则构造函数的行为将与普通函数类似。这可能会导致可怕的后果，特别是如果构造函数正在执行有权限的操作。

审计结果 ： 【通过】

安全建议 ： 无

未检验返回值

漏洞描述

在 Solidity 中存在三种向一个地址发送 Ether 的方法：transfer(), send(), call.value()。他们的区别在于 transfer 函数发送失败时会抛出异常 throw，将交易状态回滚，花费 2300gas；send 函数发送失败时返回 false，花费 2300gas；call.value 方法发送失败时返回 false，调用花费全部 gas，将导致重入攻击风险。如果在合约代码中使用 send 或者 call.value 方法进行 Ether 发送时未检查方法返回值，如果发生错误时，合约会继续执行后面得代码，将导致以为的结果。

审计结果 ： 【通过】

安全建议 ： 无

不安全的随机数

漏洞描述

以太坊区块链上的所有交易都是确定性的状态转换操作，没有不确定性，这最终意味着在区块链生态系统内不存在熵或随机性的来源。所以咋 Solidity 中没有 rand() 这种随机数功能。很多开发者使用未来的块变量，如区块哈希值，时间戳，区块高低或是 Gas 上限等来生成随机数，这些量都是由挖矿的矿工控制的，因此并不是真正随机的，因此使用过去或现在的区块变量产生随机数可能导致破坏性漏洞。

审计结果 ： 【通过】

安全建议 ： 无

时间戳依赖

漏洞描述

在以太坊区块链中，数据块时间戳（block.timestamp）被用于各种应用，例如随机数的函数，锁定一段时间的资金以及时间相关的各种状态变化的条件语句。矿工有能力根据需求调整时间戳，比如 block.timestamp 或者别名 now 可以由矿工操纵。如果在智能合约中使用错误的块时间戳，这可能会导致严重漏洞。如果合约不是特别关心矿工对区块时间戳的操纵，这可能是不必要的，但是在开发合约时应该注意这一点。

审计结果 ： 【通过】

安全建议 ： 无

交易顺序依赖

漏洞描述

在以太坊区块链中，矿工会选择来自该矿池的哪些交易将包含在该区块中，这通常是由 gasPrice 交易决定的，矿工将选择交易费最高的交易打包进区块。由于区块中的交易信息对外公开，攻击者可以观察事务池中是否存在可能包含问题解决方案的事务，修改或撤销攻击者的权限或更改合约中的对攻击者不利的状态。然后，攻击者可以从这个事务中获取数据，并创建一个更高级别的事务 gasPrice 并在原始之前将其交易包含在一个区块中，这样将抢占原始事务解决方案。

审计结果 ：【通过】

安全建议 ： 无

Delegatecall 函数调用

漏洞描述

在 Solidity 中，delegatecall 函数时标准消息调用方法，但在目标地址中的代码会在调用合约的环境下运行，也就是说，保持 msg.sender 和 msg.value 不变。该功能支持实现库，开发人员可以为未来的合约创建可重用的代码。库中的代码本身可以是安全的，无漏洞的，但是当在另一个应用的环境中运行时，可能会出现新的漏洞，所以使用 delegatecall 函数时可能会导致意外的代码执行。

审计结果 ： 【通过】

安全建议 ： 无

Call 函数调用

漏洞描述

Call 函数跟 delegatecall 函数相似，都是以太坊智能合约编写语言 Solidity 提供的底层函数，用来与外部合约或者库进行交互，但是用 call 函数方法来处理对合约的外部标准信息调用（Standard Message Call）时，代码在外部合约/功能的环境中运行。 此类函数使用时需要对调用参数的安全性进行判定，建议谨慎使用，攻击者可以很容易地借用当前合约的身份来进行其他恶意操作，导致严重漏洞。

审计结果 ： 【通过】

安全建议 ： 无

拒绝服务【低危】

漏洞描述

拒绝服务攻击的原因类别比较广泛，其目的就是让用户在一段时间内或永久地在某些情况下使合约无法正常运行，包括在作为交易接收方时的恶意行为，人为增加计算功能所需 gas 导致 gas 耗尽（比如控制 for 循环中的变量大小），滥用访问控制访问合约的 private 组件，在合约中拥有特权的 owner 被修改，基于外部调用的进展状态，利用混淆和疏忽等都能导致拒绝服务攻击。

审计结果 ：【低危】

通过审计，发现 Meer-Destory 合约中的 confirmBatchTxid 函数存在拒绝服务攻击漏洞，从代码规范来讲，无限制循环将导致 dos 拒绝服务漏洞，由于这里是 owner 操作，所以循环次数可限制可不限制，固漏洞威胁降至低危。

代码如下：

```
function confirmBatchTxid( address[] memory _senders, bytes32[] memory txId, u
int[] memory meerNum  ) public only(owner) {

        for ( uint i =0; i < _senders.length; i++ ) {

              confirmTxid( _senders[i], txId[i], meerNum[i] );
        }
    }
```

安全建议 ：

合约不应该循环通过可以被外部用户人为操纵的数据结构

逻辑设计缺陷【低危】

漏洞描述

在智能合约中，开发者为自己的合约设计的特殊功能意在稳固代币的市值或者项目的寿命，增加项目的亮点，然而越复杂的系统越容易有出错的可能，正是在这些逻辑和功能中，一个细微的失误就可能导致整个逻辑与预想出现严重的偏差，留下致命的隐患，比如逻辑判断错误，功能实现与设计不符，无限制批量刷数据等。

审计结果 ： 【低危】

通过审计，发现 Meer-Destory 合约中的 fetchMeer 函数存在批量添加垃圾数据的缺陷，任意用户都可以随意给 burnlist 中添加 redeemPublicHash 值，但是如果仅仅添加 redeemPublicHash 值，而并未进行销毁操作，将产生垃圾数据。

代码如下：

```
    function fetchMeer( bytes20 _meerPKH ) public {

    // require(burnList[msg.sender].redeemPublicHash != 0x0);
        burnList[msg.sender].redeemPublicHash = _meerPKH;
        emit FetchMeer( msg.sender, _meerPKH );
    }
```

安全建议 ：

应当将设置 redeemPublicHash 值的操作，放在销毁 burn 函数中，而且只有销毁这或者合约拥有者进行数据添加操作。

假充值漏洞

漏洞描述

在以太坊代币交易回条状态是成功还是失败（true or false），取决于交易事务执行过程中是否抛出了异常（比如使用了 require/assert/revert/throw 等机制）。当用户调用代币合约的 transfer 函数进行转账时，如果 transfer 函数正常运行未抛出异常，无聊转账交易是否成功，该交易的回执状态就是成功即 true。那么有些代币合约的 transfer 函数对转账发起人(msg.sender)的余额检查用的是 if 判断方式，当 balances[msg.sender] < _value 时进入 else 逻辑部分并 return false，最终没有抛出异常，但是交易回执是成功的，那么我们认为仅 if/else 这种温和的判断方式在 transfer 这类敏感函数场景中是一种不严谨的编码方式，将导致相关中心化交易所、中心化钱包、代币合约的假充值漏洞。

审计结果 ： **【通过】**

安全建议 ： **无**

短地址攻击漏洞

漏洞描述

在 Solidity 智能合约中，将参数传递给智能合约时，参数将根据 ABI 规范进行编码。EVM 运行攻击者发送比预期参数长度短的编码参数。例如在交易所或者钱包转账时，需要发送转账地址 address 和转账金额 value，攻击者可以发送 19 字节的地址而不是标准的 20 字节地址，在这种情况下，EVM 会将 0 填到编码参数的末尾以补成预期的长度，这将导致最后转账金额参数 value 的溢出，从而改变原本转账金额。

审计结果 ： 【通过】

安全建议 ： 无

未初始化的存储指针

漏洞描述

以太坊 EVM 既用 storage 来存储变量，也用 memory 来存储变量，函数内的局部变量根据它们的类型默认用 storage 或 memory 存储，在 Solidity 的工作方式里面，状态变量按它们出现在合约中的顺序存储在合约的 Slot 中 ，未初始化的局部 storage 变量可能会指向合约中的其他意外存储变量，从而导致有意或无意的漏洞。

审计结果 ： 【通过】

安全建议 ： 无

代币增发

漏洞描述

在合约部署完成初始化发行代币总量确定后，检测合约代码中是否存在可修改代币发行总数的逻辑功能，如果存在修改代币发行总量的功能接口时，此功能接口是否存在正确的权限验证。

审计结果 ： 【通过】

安全建议 ： 无

冻结账户绕过

漏洞描述

在合约中的转账操作代码中，检测合约代码中是否存在对转账账户冻结状态检查的逻辑功能，如果转账账户已经冻结，是否可被绕过的风险。

审计结果 ： 【通过】

安全建议 ： 无

附录 A：漏洞风险等级评估标准

漏洞风险等级评估标准	
漏洞等级	漏洞风险描述
高危	能直接导致代币合约或者用户数字资产损失的漏洞，比如：整数溢出漏洞、假充值漏洞、重入漏洞、代币违规增发等。 能直接造成代币合约所属权变更或者验证绕过的漏洞，比如：权限验证绕过、call 代码注入、变量覆盖、未验证返回值等。 能直接导致代币正常工作的漏洞，比如：拒绝服务漏洞、不安全的随机数等。
中危	需要一定条件才能出发的漏洞，比如代币所有者高权限触发的漏洞，交易顺序依赖漏洞等。 不能直接造成资产损失的漏洞，比如函数默认可见性错误漏洞，逻辑设计缺陷漏洞等。
低危	难以触发的漏洞，或者不能导致资产损失的漏洞，比如需要高于攻击收益的代价才能触发的漏洞 无法导致安全漏洞的错误编码问题。

附录 B：漏洞测试工具

安全审计工具	
公开审计工具	https://audit.noneage.com
内部审计工具	内部审计工具

附录 C：智能合约安全审计危险等级说明

web 网站安全测试等级说明	
安全等级	等级说明
★★★★★	0 < 危险等级≤2
★★★★☆	2 < 危险等级≤4
★★★☆☆	4 < 危险等级≤6
★★☆☆☆	6 < 危险等级≤8
★☆☆☆☆	8 < 危险等级≤10

危险等级计算方法：

$$危险等级 = \frac{漏洞1 \times 危险分值 + 漏洞2 \times 危险分值 + \cdots + 漏洞n \times 危险分值}{sum（漏洞1 + 漏洞2 + \cdots + 漏洞n）}$$

漏洞危险分值说明	
漏洞等级	危险分数范围
高危	7 < 危险分数≤10
中危	4 < 危险分数≤7
低危	0 < 危险分数≤4

注：具体分数由零时科技安全专业根据实际情况讨论确定

零时科技

微信扫描二维码，关注我的公众号

联系电话：17391945345
17391948456
邮箱地址：support@noneage.com
网站网址：https://www.noneage.com
联系地址：深圳市深南大道佳嘉豪商务大厦 18A

深圳零时科技有限公司