智能合约安全审计报告

Cherryswap 智能合约安全审计报告

审计团队：零时科技安全团队

时间： 2021-07-09

Cherryswap 智能合约安全审计报告

1.概述

零时科技安全团队于 2021 年 07 月 05 日，接到 **Cherryswap 项目**的安全审计需求，团队于 2021 年 07 月 09 日对 **Cherryswap 智能合约**进行了安全审计，审计过程中零时科技安全审计专家与 Cherryswap 项目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审计工作，尽量规避在测试过程中对项目产生和运营造成风险。

经过与 Cherryswap 项目方沟通反馈确认审计过程中发现的漏洞及风险均已修复或在可承受范围内，本次 Cherryswap 智能合约安全审计结果：通过安全审计。

合约报告 MD5：7D38BEA3BF996201E4E247A03A0DAE79

2.项目背景

2.1 项目简介

项目名称：Cherryswap

项目官网：https://www.cherryswap.net/

合约类型：代币合约

代码语言：Solidity

官方 GitHub 仓库地址：https://github.com/cherryswapnet/

合约文件：IFO.sol，IFOByProxy.sol，SingleSmartChef.sol，CherryToken.sol，MasterChef.sol，SmartChef.sol，SyrupBar.sol，CherryERC20.sol，CherryFactory.sol，CherryPair.sol，CherryMigrator.sol，CherryRouter.sol，CherryRouter01.sol

2.2 审计范围

Cherryswap 官方提供合约及对应 MD5 地址：

IFO.sol	532EA1ACA92CA04C252F6C30CE757849
IFOByProxy.sol	5C1EC1C732D497926CD456A90E7CD983
SingleSmartChef.sol	1C12B3DA2A9B86E862DF0CF8345A6C33
CherryToken.sol	6C17EA78785083E8DD596E76A96C305C
MasterChef.sol	734BE75529A3C3223FECFE0037411EB2
SmartChef.sol	1258A607835B667811DE211FC9C3EA59
SyrupBar.sol	9EB0C60BAFB31FAE2AAB941446BC0E5A
CherryERC20.sol	6B497701DB11C5D36A9634CF6D0B9C29
CherryFactory.sol	B7452F76A1960BDB3DBE6323D3CDAE14
CherryPair.sol	FA8A3309A7BB36742DEC4714E867E982
CherryMigrator.sol	E0A4E2E8C39B1BBF229689F123143664
CherryRouter.sol	7B24CB0694FA99B6E5232D45F810F512
CherryRouter01.sol	ADC124088EAB7E125C450F5D15636174

3.合约架构分析

3.1 目录结构

```
└─Cherryswap
    ├─swap-core-main
    │  └─contracts
    │        CherryERC20.sol
    │        CherryFactory.sol
    │        CherryPair.sol
    │  ├─swap-farm-main
    │  └─contracts
    │        CherryToken.sol
    │        MasterChef.sol
    │        IFO.sol
    │        IFOByProxy.sol
    │        SingleSmartChef.sol
    │        SmartChef.sol
    │        SyrupBar.sol
    │  └─swap-periphery-main
    │  └─contracts
    │        CherryMigrator.sol
    │        CherryRouter.sol
    │        CherryRouter01.sol
```

3.2 合约详情

IFO Contract

方法名称	方法传参	方法属性
setOfferingAmount()	uint256 _offerAmount	onlyAdmin
setRaisingAmount()	uint256 _raisingAmount	onlyAdmin
deposit()	uint256 _amount	public
harvest()	none	public
hasHarvest()	address _user	external
getUserAllocation()	address _user	public
getOfferingAmount()	address _user	public
getRefundingAmount()	address _user	public
getAddressListLength()	none	external
finalWithdrawLpToken()	uint256 _lpAmount	onlyAdmin
finalWithdrawOfferingToken()	uint256 _offerAmount	onlyAdmin

IFOByProxy Contract

方法名称	方法传参	方法属性
initialize()	IBEP20 _lpToken,IBEP20 _offeringToken,uint256 _startBlock,uint256 _endBlock,uint256 _offeringAmount,uint256 _raisingAmount,address _adminAddress	initializer
setOfferingAmount()	uint256 _offerAmount	onlyAdmin
setRaisingAmount()	uint256 _raisingAmount	onlyAdmin
deposit()	uint256 _amount	public
harvest()	none	public
hasHarvest()	address _user	external
getUserAllocation()	address _user	public
getOfferingAmount()	address _user	public
getRefundingAmount()	address _user	public
getAddressListLength()	none	external
finalWithdrawLpToken()	uint256 _lpAmount	onlyAdmin
finalWithdrawOfferingToken()	uint256 _offerAmount	onlyAdmin

SingleSmartChef Contract

方法名称	方法传参	方法属性
stopReward()	none	onlyOwner
getMultiplier()	uint256 _from, uint256 _to	public
updateMultiplier()	uint256 multiplierNumber	onlyOwner
pendingReward()	address _user	external
updatePool()	uint256 _pid	public
massUpdatePools()	none	public
deposit()	uint256 _amount	public
withdraw()	uint256 _amount	public
emergencyWithdraw()	none	public
emergencyRewardWithdraw()	uint256 _amount	onlyOwner

4.审计详情

4.1 风险分布

风险名称	风险级别	修复状态
管理员权限问题	低	已确认
变量更新安全	无	正常
整数溢出	无	正常
浮点数和数值精度	无	正常
默认可见性	无	正常
tx.origin 身份认证	无	正常
错误的构造函数	无	正常
未验证返回值	无	正常
不安全的随机数	无	正常
时间戳依赖	无	正常
交易顺序依赖	无	正常
Delegatecall 函数调用	无	正常
Call 函数调用	无	正常
拒绝服务	无	正常
逻辑设计缺陷	无	正常
假充值漏洞	无	正常
短地址攻击漏洞	无	正常
未初始化的存储指针	无	正常
冻结账户绕过	无	正常
合约调用者未初始化	无	正常
重入攻击	无	正常

4.2 风险审计详情

4.2.1 管理员权限问题

- **漏洞描述**

智能合约管理员如果有较大的操作合约权限，如果该管理员被恶意人员操控，或可导致异常资金流失及市场稳定性动摇。

- **审计结果： 存在管理员权限较大问题**
- **风险代码**

Cherryswap 中 IFO 合约，该合约存在管理员转移合约中 offeringToken 操作，如果该部署者被恶意人员操控，或可导致异常资金流失及市场稳定性动摇，如下部分代码所示：

```
  function finalWithdrawOfferingToken(uint256 _offerAmount) public onlyAdmin {
    require (_offerAmount <= offeringToken.balanceOf(address(this)), 'not enough token 1');
    if(_offerAmount > 0) {
      offeringToken.safeTransfer(address(msg.sender), _offerAmount);
    }
  }
```

- **安全建议**

在保证安全的前提下，多份合理的保存私钥。

- **修复状态**

通过与 Cherryswap 团队沟通，上述功能为正常操作，且会妥善管理所有者私钥。

4.2.2 变量更新安全

- **漏洞描述**

一般来说，每个合约代码中都会存在多个变量值，有时可能会因为变量较多或者编写大意，会将某些变量不能及时更新，可能是一个代码小问题，但对有涉及资金的变量计算来说，在攻击者眼中就是可以盗取资金的大漏洞。

- **审计结果： 通过**

4.2.3 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，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**

4.2.4 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 1 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**

4.2.5 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**

4.2.6 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**

4.2.7 错误的构造函数

- **漏洞描述**

在 solidity 智能合约中的 0.4.22 版本之前，所有的合约和构造函数同名。编写合约时，如果构造函数名和合约名不相同，合约会添加一个默认的构造函数，自己设置的构造函数就会被当做普通函数，导致自己原本的合约设置未按照预期执行，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**

4.2.8 未检验返回值

- **漏洞描述**

在 Solidity 中存在三种向一个地址发送代币的方法：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 方法进行代币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**

4.2.9 不安全的随机数

- **漏洞描述**

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**

4.2.10 时间戳依赖

- **漏洞描述**

在区块链中，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**

4.2.11 交易顺序依赖

- **漏洞描述**

在区块链中，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**

4.2.12 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**

4.2.13 Call 函数调用

- **漏洞描述**

Call 函数跟 delegatecall 函数相似，都是智能合约编写语言 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**

4.2.14 拒绝服务

- **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： 通过**

4.2.15 逻辑设计缺陷

- **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等。

- **审计结果： 通过**

4.2.16 假充值漏洞

- **漏洞描述**

在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**

4.2.17 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**

4.2.18 未初始化的存储指针

- **漏洞描述**

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**

4.2.19 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**

4.2.20 合约调用者未初始化

- **漏洞描述**

在合约中的 initialize 函数可被其他攻击者抢在 owner 之前调用，从而初始化管理员地址。

- **审计结果： 通过**

4.2.21 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 代币时，它将调用恶意代码，Solidity 中的 call.value()函数在被用来发送代币时会消耗他接收到的所有 gas，所以当调用 call.value()函数发送代币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**

5.安全审计工具

工具名称	功能
Oyente	可以用来检测智能合约中常见 bug
securify	可以验证以太坊智能合约的常见类型
MAIAN	可以查找多个智能合约漏洞并进行分类
零时内部工具包	零时(鹰眼系统)自研发工具包+https://audit.noneage.com

免责声明：

零时科技仅就本报告出具之前发生或存在的事实出具报告并承担相应责任，对于出具报告之后发生的事实由于无法判断智能合约安全状态，因此不对此承担责任。零时科技对该项目约定内的安全审计项进行安全审计，不对该项目背景及其他情况进行负责，项目方后续的链上部署以及运营方式不在本次审计范围。本报告只基于信息提供者截止出具报告时向零时科技提供的信息进行安全审计，对于此项目的信息有隐瞒，或反映的情况与实际情况不符的，零时科技对由此而导致的损失和不利影响不承担任何责任。

市场有风险，投资需谨慎，此报告仅对智能合约代码进行安全审计和结果公示，不作投资建议和依据。

零时科技
NONEAGE

邮　　箱：support@noneage.com
官　　网：www.noneage.com
微　　博：weibo.com/noneage