智能合约安全审计报告

Powerswap 智能合约安全审计报告

审计团队：零时科技安全团队

时间： 2021-03-31

Powerswap智能合约安全审计报告

1.概述

零时科技安全团队于2021年02月24日，接到 **Powerswap项目**的安全审计需求，团队与2021年03月22日至2021年03月31日对 **Powerswap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Powerswap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Powerswap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Powerswap智能合约安全审计结果：通过安全审计。

合约报告MD5： 1667E0E9868DC236C505AFFCA51EB3B7

2.项目背景

2.1 项目简介

> 项目名称：Powerswap
>
> 项目官网：https://ht.powerswap.org/
>
> 合约类型：代币合约
>
> 代码语言： Solidity
>
> 合约文件： PowerswapFactory.sol、PowerswapFactory.sol、WETH9.sol

2.2 审计范围

Powerswap官方提供链上合约地址：

PowerswapFactory https://cn.etherscan.com/address/0xB89658d9636744D0b016b4AC0d71935d667c2065#code

PowerswapRouter https://cn.etherscan.com/address/0xe03B41ccB844C277616e5AB7F25eCfB99Ad99917#code

WETH9 https://cn.etherscan.com/address/0x31Ea6627C958531823D50024028dF6635DD3faE5#code

2.3 安全审计项

零时科技安全专家对约定内的安全审计项目进行安全审计，本次智能合约安全审计的范围，不包含未来可能出现的新型攻击方式，不包含合约升级活篡改后的代码，不包括在后续跨链部署时的操作过程，不包含项目前端代码安全与项目平台服务器安全。

本次智能合约安全审计项目包括如下：

- 整数溢出
- 重入攻击
- 浮点数和数值精度
- 默认可见性
- Tx.origin身份验证
- 错误的构造函数
- 未验证返回值
- 不安全的随机数
- 时间戳依赖
- 交易顺序依赖
- Delegatecall调用
- Call调用
- 拒绝服务
- 逻辑设计缺陷
- 假充值漏洞
- 短地址攻击
- 未初始化的存储指针
- 代币增发
- 冻结账户绕过
- 权限控制
- Gas使用

3.合约架构分析

3.1 目录结构

```
|
└─Powerswap
    PowerswapFactory.sol
    PowerswapRouter.sol
    WETH9.sol
```

3.2 Powerswap 合约

Contract

PowerswapERC20

- _mint(address to, uint value)
- _burn(address from, uint value)
- _approve(address owner, address spender, uint value)
- _transfer(address from, address to, uint value)
- approve(address spender, uint value)
- transfer(address to, uint value)
- transferFrom(address from, address to, uint value)
- permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s)

PowerswapPair

- getReserves()
- getBalances()
- getPrices()
- getR()
- _safeTransfer(address token, address to, uint value)
- initialize(address _token0, address _token1)
- setR(uint16 _R)
- _updateAtTheInitialBlock(uint balance0, uint balance1)
- _updateAtTheFirstCallInEachBlock(uint112 _reserve0, uint112 _reserve1, uint32 timeElapsed)
- _update(uint balance0, uint balance1, uint112 _reserve0, uint112 _reserve1)
- _updateAfterSwap
- _mintFee(uint112 _reserve0, uint112 _reserve1)
- mint(address to)
- burn(address to)
- swap(uint amount0Out, uint amount1Out, address to, bytes calldata data)
- updateVirtualBalances()
- skim(address to)
- sync()

PowerswapFactory

- allPairsLength()
- createPair(address tokenA, address tokenB)
- setFeeTo(address _feeTo)
- setFeeToSetter(address _feeToSetter)
- setrSetter(address _rSetter)

PowerswapRouter

- _addLiquidity(address tokenA,address tokenB,uint amountADesired,uint amountBDesired,uint amountAmin,uint amountBmin)
- addLiquidity(address tokenA,address tokenB,uint amountADesired,uint amountBDesired,uint amountAmin,uint amountBmin,address to,uint deadline)
- addLiquidityETH(address token,uint amountTokenDesired,uint amountTokenMin,uint amountETHMin,address to,uint deadline)
- removeLiquidity(address tokenA,address tokenB,uint liquidity,uint amountAMin,uint amountBMin,address to,uint deadline)
- removeLiquidityETH(address token,uint liquidity,uint amountTokenMin, uint amountETHMin,address to,uint deadline)
- _removeLiquidityWithPermit(address tokenA,address tokenB,uint liquidity,uint amountAMin, uint amountBMin,address to,uint deadline,bool approveMax, uint8 v, bytes32 r, bytes32 s)
- removeLiquidityWithPermit(address tokenA,address tokenB,uint liquidity,uint amountAMin,uint amountBMin,address to,uint deadline,bool approveMax, uint8 v, bytes32 r, bytes32 s)
- removeLiquidityETHWithPermit(address token,uint liquidity,uint amountTokenMin,uint amountETHMin,address to,uint deadline, bool approveMax, uint8 v, bytes32 r, bytes32 s)

- removeLiquidityETHSupportingFeeOnTransferTokens(address token,uint liquidity,uint amountTokenMin,uint amountETHMin,address to,uint deadline)
- removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(address token,uint liquidity,uint amountTokenMin,uint amountETHMin, address to, uint deadline,bool approveMax, uint8 v, bytes32 r, bytes32 s)
- function _swap(uint[] memory amounts, address[] memory path, address _to)
- function swapExactTokensForTokens(uint amountIn,uint amountOutMin,address[] calldata path,address to,uint deadline)
- swapTokensForExactTokens(uint amountOut,uint amountInMax,address[] calldata path,address to,uint deadline)
- swapExactETHForTokens(uint amountOutMin,address[] calldata path,address to,uint deadline)
- swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address to, uint deadline)
- swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address to, uint deadline)
- swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)
- _swapSupportingFeeOnTransferTokens(address[] memory path, address _to)
- swapExactTokensForTokensSupportingFeeOnTransferTokens(uint amountIn,uint amountOutMin,address[] calldata path,address to,uint deadline)
- swapExactETHForTokensSupportingFeeOnTransferTokens(uint amountOutMin,address[] calldata path,address to,uint deadline)
- swapExactTokensForETHSupportingFeeOnTransferTokens(uint amountIn, uint amountOutMin, address[] calldata path, address to, uint deadline)
- quote(uint amountA, uint reserveA, uint reserveB)
- getAmountOut(uint amountIn, uint reserveIn, uint reserveOut, uint price, uint16 R)
- getAmountIn(uint amountOut, uint reserveIn, uint reserveOut, uint price, uint16 R)
- getAmountsOut(uint amountIn, address[] memory path)
- getAmountsIn(uint amountOut, address[] memory path)

WETH9

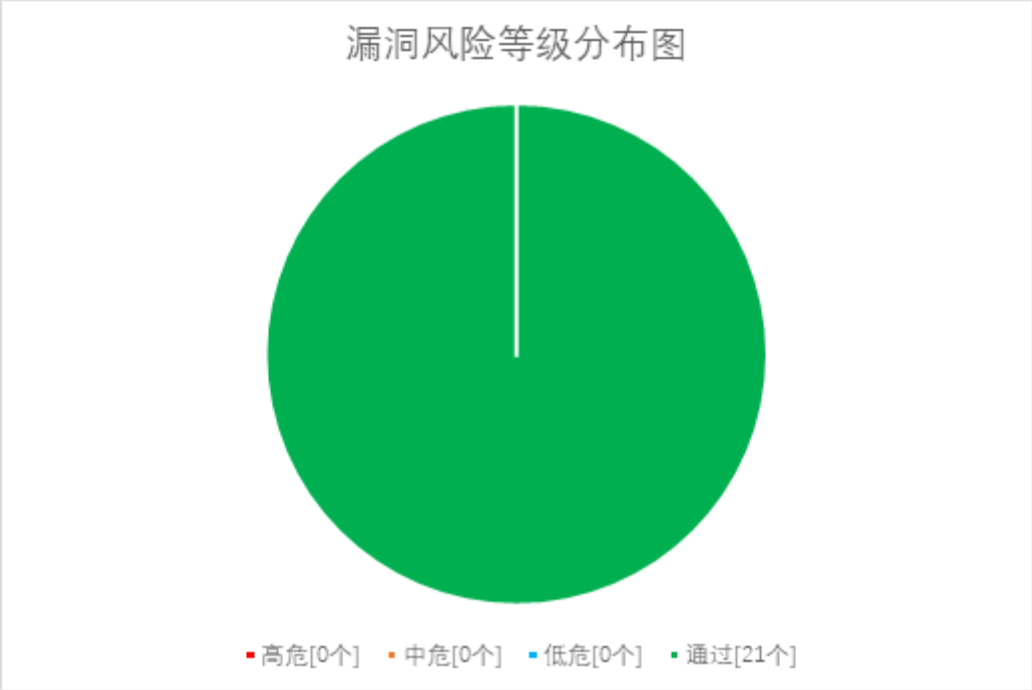
- deposit()
- withdraw(uint wad)
- totalSupply()
- approve(address guy, uint wad)
- mint(uint _value)
- transfer(address dst, uint wad)
- transferFrom(address src, address dst, uint wad)

4.审计详情

4.1 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	0	21



本次智能合约安全审计高危漏洞0个，中危0个，低危0个，通过21个，安全等级高。

4.2 漏洞详情

对约定内的智能合约进行安全审计，未发现可以直接利用并产生安全问题的安全漏洞，通过安全审计。

4.3 其他风险

其他风险是指合约安全审计人员认为有风险的代码，在特定情况下可能会影响项目稳定性，但不能构成直接危害的安全问题。

4.3.1 增发问题

- **问题点**

WETH9合约中，存在mint()函数，经过调试，该合约存在增发问题，代码如下所示：

```
function mint(uint _value) public payable{
    balanceOf[msg.sender] += _value;
    Deposit(msg.sender, _value);
}
```

- **安全建议**

建议及时删除敏感函数。

- **修复状态**

通过与Powerswap团队沟通，WETH9合约不是用来发币，mint()函数无影响。

4.3.2 安全函数使用

- **问题点**

WETH9合约中，存币deposit()函数，铸币mint()函数，转账transferFrom()函数都存在算术运算，并且均未使用安全函数，存在整数溢出的风险，如下代码所示：

```
    function deposit() public payable {
        balanceOf[msg.sender] += msg.value;
        emit Deposit(msg.sender, msg.value);
    }
    function mint(uint _value)public payable{
        balanceOf[msg.sender] += _value;
        Deposit(msg.sender, _value);
    }
    function transferFrom(address src, address dst, uint wad) public returns
(bool){
        require(balanceOf[src] >= wad, "");
        if (src != msg.sender && allowance[src][msg.sender] != uint(-1)) {
            require(allowance[src][msg.sender] >= wad, "");
            allowance[src][msg.sender] -= wad;
        }
        balanceOf[src] -= wad;
        balanceOf[dst] += wad;
        emit Transfer(src, dst, wad);
        return true;
    }
```

- **安全建议**

为了排除整数溢出的可能性，建议在算术运算时尽量使用安全函数safemath。

- **修复状态**

通过与Powerswap团队沟通，新函数已经更新，包含safemath的功能。

4.3.3 地址有效性校验

- **问题点**

WETH9合约中，授权函数approve()和转账函数transfer()均有地址参数传入，并未对传入的地址进行有效性验证，如果传入的地址参数为0，可能会导致转币丢失等安全问题。

```
    function approve(address guy, uint wad) public returns (bool) {
        allowance[msg.sender][guy] = wad;
        emit Approval(msg.sender, guy, wad);
        return true;
    }
    function transfer(address dst, uint wad) public returns (bool) {
        return transferFrom(msg.sender, dst, wad);
    }
```

- **安全建议**

添加地址有效性校验，如下代码所示：

```
require(Address != address(0) , 'ADDRESS ERROR!!!');
```

- **修复状态**

通过与Powerswap团队沟通，该地址参数无影响。

4.3.4 算数问题一

- **问题点**

PowerswapPair合约，_updateAfterSwap()函数中变量加减操作并未使用安全函数，如下代码所示：

```
    function _updateAfterSwap(
        uint amount0In,  uint amount1In,
        uint amount0Out, uint amount1Out,
        uint balance0,   uint balance1) private {
        if(amount0In > 0){
        balanceBuy0 += amount0In;
        balanceBuy1 -= amount1Out;
        }else{
        balanceSell0 -= amount0Out;
        balanceSell1 += amount1In;
        }
        reserve0 = uint112(balance0);
        reserve1 = uint112(balance1);
        emit Sync(reserve0, reserve1);
    }
```

- **安全建议**

建议使用智能合约函数库中的 SafeMath 来处理算术逻辑。

- **修复状态**

通过与Powerswap团队沟通，加法运算是两个uint112相加，不存在溢出；减法运算时有溢出会报错，也就不存在安全风险。

4.3.5 算数问题二

- **问题点**

PowerswapPair合约，_updateAtTheFirstCallInEachBlock()函数中变量加减操作并未使用安全函数，如下代码所示：

```
function _updateAtTheFirstCallInEachBlock(uint112 _reserve0, uint112 _reserve1,
uint32 timeElapsed) private{
        balanceBuy0  = _reserve0;
        balanceBuy1  = _reserve1;
        balanceSell0 = _reserve0;
        balanceSell1 = _reserve1;
        uint _priceBuy  = priceBuy;
        uint _priceSell = priceSell;
        uint16 _R = R;
        priceBuy  = (_priceBuy.mul(_reserve1).mul(_R).add((uint(_reserve0) <<
RESOLUTION).mul(65535-_R)))/(_reserve1.mul(65535));
        priceSell = (_priceSell.mul(_reserve0).mul(_R).add((uint(_reserve1) <<
RESOLUTION).mul(65535-_R)))/(_reserve0.mul(65535));
        price0CumulativeLast +=
uint(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) * timeElapsed;
        price1CumulativeLast +=
uint(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) * timeElapsed;
    }
```

- **安全建议**

建议使用智能合约函数库中的 SafeMath 来处理算术逻辑。

- **修复状态**

通过与Powerswap团队沟通，最终运算结果并未进行其他操作，不产生影响。

4.3.6 管理员权限过大

- **问题点**

PowerswapPair合约中，factory管理员可以定义初始化token地址，也可以设置R值，如果该管理员地址私钥被盗或者被恶意人员操控，就可以重新利用初始化函数定义token地址，或者设置R值，导致出现非常规的数字货币获取及动摇市场稳定性，如下代码所示：

```
    function initialize(address _token0, address _token1) external override{
        require(msg.sender == factory, 'Powerswap: FORBIDDEN');
        token0 = _token0;
        token1 = _token1;
        R = 32767;  // default r = 0.5;
    }
    function setR(uint16 _R) external override {
        address _rSetter = IPowerswapFactory(factory).rSetter();
        require(msg.sender == _rSetter, 'Powerswap: FORBIDDEN');
        R = _R;
    }
```

- **安全建议**

多份合理的保存私钥；可设置初始化token地址只调用一次。

- **修复状态**

通过与Powerswap团队沟通，团队表示会妥善管理私钥。

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

6.漏洞风险评估标准

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	需要一定条件才能触发的漏洞，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。

零时科技
NONEAGE

咨询电话：86-17391945345 18511993344
邮　　箱：support@noneage.com
官　　网：www.noneage.com
微　　博：weibo.com/noneage