智能合约安全审计报告

Hexch 智能合约安全审计报告

审计团队：零时科技安全团队

时间： 2021-04-08

Hexch智能合约安全审计报告

1.概述

零时科技安全团队于2021年04月07日，接到 **Hexch项目**的安全审计需求，团队与2021年04月07日至2021年04月08日对 **Hexch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Hexch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Hexch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Hexch智能合约安全审计结果：通过安全审计。

合约报告MD5：D37214BFDC483F0715A0E15395D3BFDA

2.项目背景

2.1 项目简介

> 项目名称：Hexch
>
> 项目官网：https://hexch.io
>
> 合约类型：代币合约
>
> 代码语言： Solidity
>
> 部署公链：Huobi ECO Chain
>
> 合约地址： 0xe3ffa0ca53f8729daf54b866bd34a6c132e657a3

2.2 审计范围

Hexch官方链上合约：https://hecoinfo.com/address/0xe3ffa0ca53f8729daf54b866bd34a6c132e657a3#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

└─hexch.sol

3.2 AuctionPool合约

Contract

swapV1

- safeTransfer(address token, address to, uint256 value)
- safeTransferFrom(address token, address from, address to, uint value)
- pairFor(address factory, address tokenA, address tokenB, bytes memory initCode)
- addRouter(address router)
- isRouter(address router)
- addFactory(address _factory, uint256 fee, bytes memory initCode)
- setFeeFlag(uint256 f)
- setFeeRate(uint256 fee)
- _needFee()
- callExProxy(address router, IERC20 inToken, IERC20 outToken, uint256 amountIn, uint256 amountOutMin, bytes memory data)
- swapExactTokensForTokens(address factory, IERC20 inToken, IERC20 outToken, uint256 amountIn, uint256 amountOutMin, uint deadline, address[] memory path)
- swapTokensForExactTokens(address factory, IERC20 inToken, IERC20 outToken, uint256 amountInMax, uint256 amountOut, uint deadline, address[] memory path)
- transferFromUser(IERC20 erc, address _from, uint256 _inAmount)
- approve(IERC20 erc, address approvee)

- viewBalance(IERC20 erc, address owner)
- sendFunds(IERC20 erc, address payable receiver, uint256 funds)
- _swap(address factory, uint[] memory amounts, address[] memory path, address _to, bytes memory initCode)
- withdrawEth()
- withdrawAnyToken(IERC20 erc)
- exitContract()
- _getMiningReward(address addr)

IWETH

- balanceOf(address account)
- deposit()
- transfer(address to, uint value)
- withdraw(uint)

Interface

IUniswapV2Pair

- token0()
- token1()
- getReserves()
- swap(uint amount0Out, uint amount1Out, address to, bytes calldata data)

IERC20

- balanceOf(address account)
- transfer(address recipient, uint256 amount)
- allowance(address owner, address spender)
- approve(address spender, uint256 amount)
- transferFrom(address sender, address recipient, uint256 amount)

IMdexSwapMining

- takerWithdraw()

Library

UniswapV2Library

- sortTokens(address tokenA, address tokenB)
- pairFor(address factory, address tokenA, address tokenB, bytes memory initCode)
- getReserves(address factory, address tokenA, address tokenB, bytes memory initCode)
- getAmountOut(uint amountIn, uint reserveIn, uint reserveOut, uint fee)
- getAmountIn(uint amountOut, uint reserveIn, uint reserveOut, uint fee)
- getAmountsOut(address factory, uint amountIn, address[] memory path, bytes memory initCode, uint fee)
- getAmountsIn(address factory, uint amountOut, address[] memory path, bytes memory initCode, uint fee)

SafeERC20

- safeTransfer(IERC20 token, address to, uint256 value)
- safeTransferFrom(IERC20 token, address from, address to, uint256 value)
- safeApprove(IERC20 token, address spender, uint256 value)
- callOptionalReturn(IERC20 token, bytes memory data)

Address

- isContract(address account)

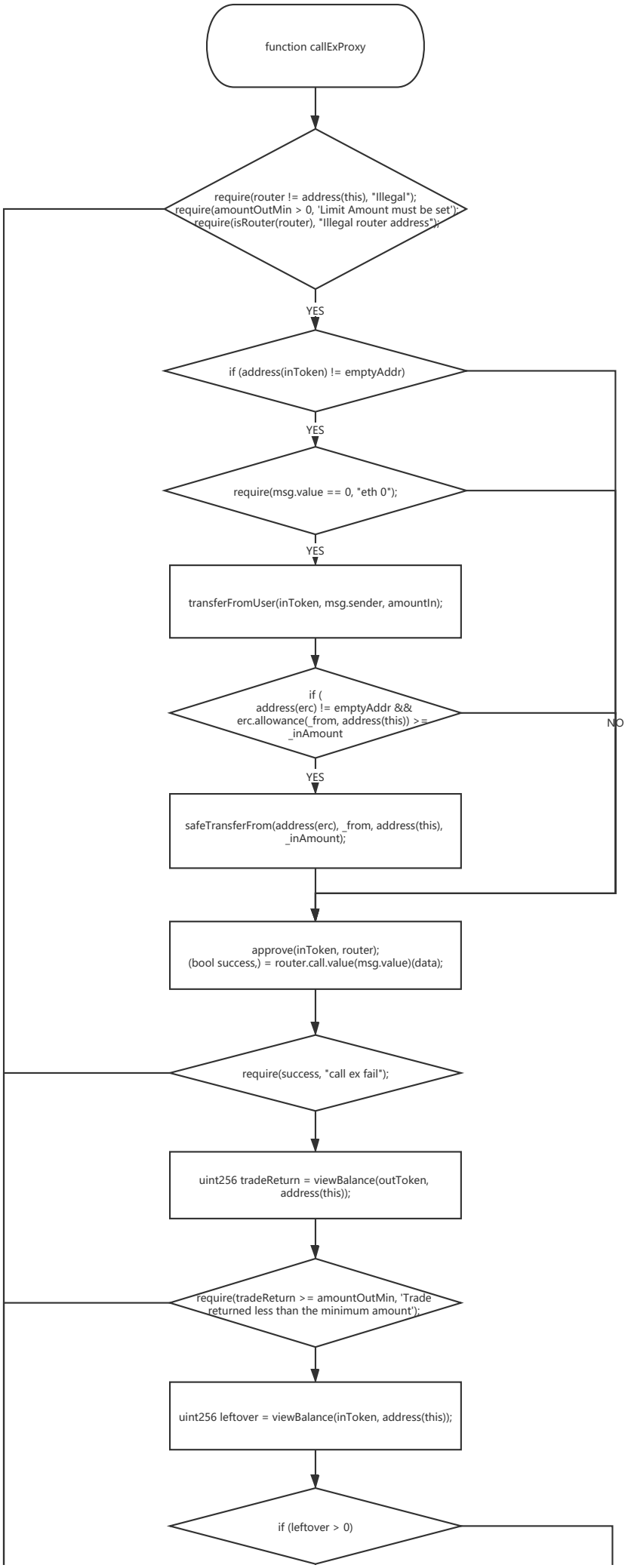
SafeMath

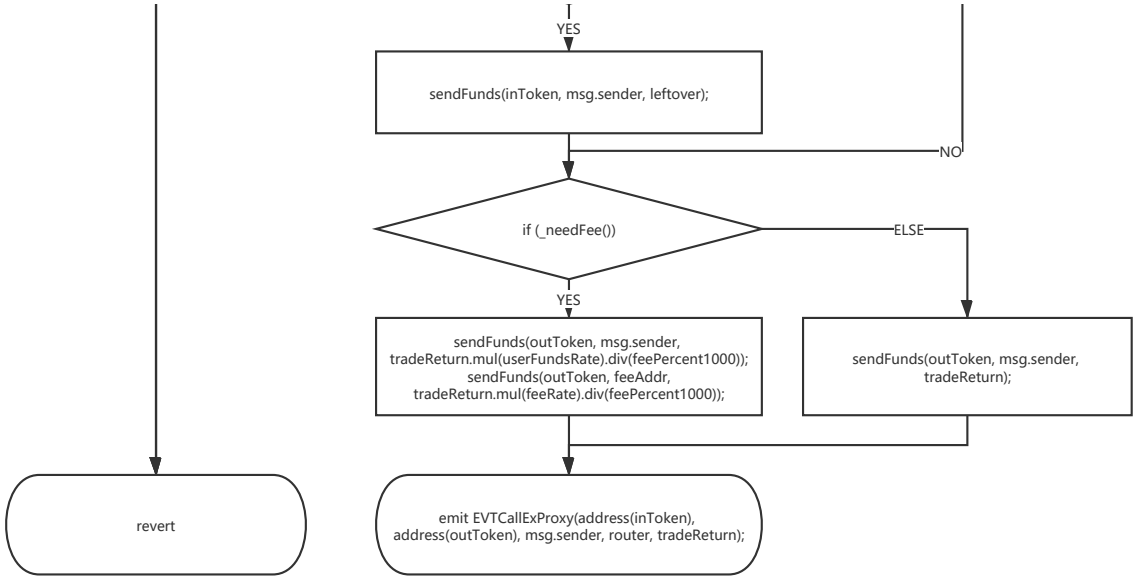
- add(uint256 a, uint256 b)
- sub(uint256 a, uint256 b)
- mul(uint256 a, uint256 b)
- div(uint256 a, uint256 b)
- mod(uint256 a, uint256 b)

3.5 合约部分逻辑流程图

通过对**Hexch合约**的安全审计，安全审计人员列出了审计过程中部分合约逻辑的代码流程图，如下：

swapV1合约部分逻辑：callExProxy()


function callExProxy
require(router != address(this), "Illegal");
require(amountOutMin > 0, 'Limit Amount must be set');
require(isRouter(router), "Illegal router address");
YES
if (address(inToken) != emptyAddr)
YES
require(msg.value == 0, "eth 0");
YES
transferFromUser(inToken, msg.sender, amountIn);
if (
address(erc) != emptyAddr &&
erc.allowance(_from, address(this)) >=
_inAmount
NO
YES
safeTransferFrom(address(erc), _from, address(this),
_inAmount);
approve(inToken, router);
(bool success,) = router.call.value(msg.value)(data);
require(success, "call ex fail");
uint256 tradeReturn = viewBalance(outToken,
address(this));
require(tradeReturn >= amountOutMin, 'Trade
returned less than the minimum amount');
uint256 leftover = viewBalance(inToken, address(this));
if (leftover > 0)

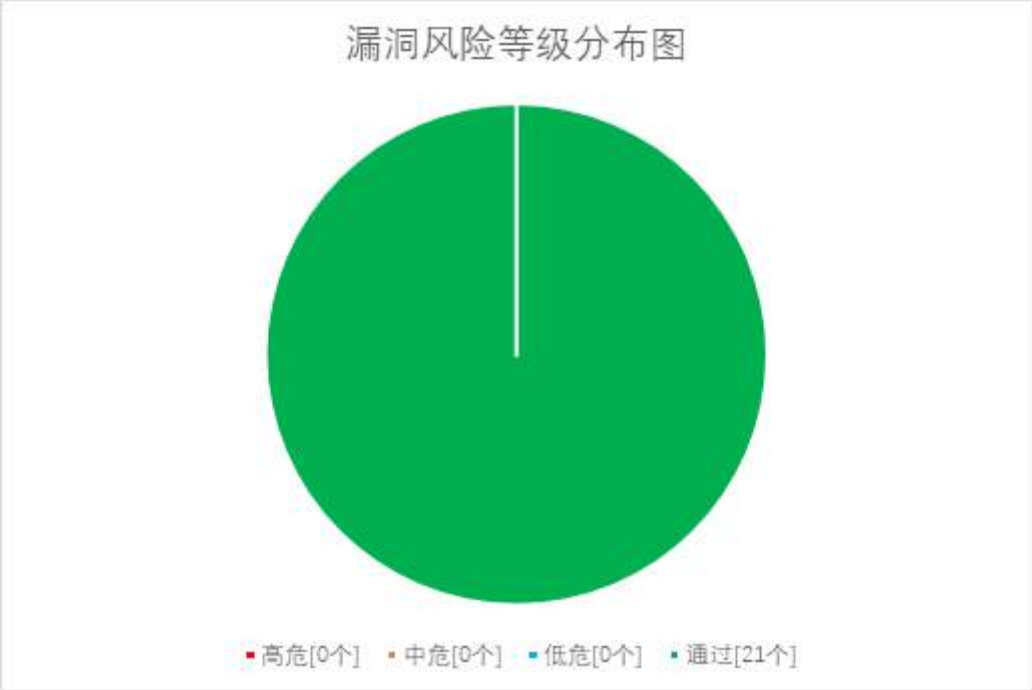


4.审计详情

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 管理员权限过大问题

- **发生原因**

智能合约中的管理员权限过大，当管理员密钥丢失或者被恶意人员控制，就会影响该项目的正常运行。

- **问题点**

通过审计合约发现swapV1合约中存在多处设置，转账和自毁均由管理员直接或者间接控制，如果管理员密钥丢失或者被恶意人员控制，就会导致该项目无法正常运行或者该合约销毁，具体代码如下：

```
function addRouter(address router) public onlyOwner {
    routerMap[router] = true;
}

function addFactory(address _factory, uint256 fee, bytes memory initCode) public
onlyOwner {
    factoryMap[_factory] = initCode;
    factoryFeeMap[_factory] = fee;
}

function setFeeFlag(uint256 f) public onlyOwner {
    feeFlag = f;
}

function setFeeRate(uint256 fee) public onlyOwner {
    require(fee > 0 && fee <= 10, "1-10");
    feeRate = fee;
}

function withdrawEth() external onlyOwner {
    msg.sender.transfer(address(this).balance);
}

function withdrawAnyToken(IERC20 erc) external onlyOwner {
    safeTransfer(address(erc), msg.sender,erc.balanceOf(address(this)));
}

function exitContract() public onlyOwner {
    selfdestruct(msg.sender);
}
```

- **安全建议**

可使用时间锁防止代币被大量转币；需安全有效存储部署者地址私钥，避免丢失或者被恶意人员获取。

4.3.2 授权过大问题

- **发生原因**

管理员在授权参数或者地址时，对该值赋予了较大的权限，如果被赋予的地址对合约进行危险操作，则会影响该项目稳定性。

- **问题点**

通过审计合约发现swapV1合约中存在授权过大问题，如approve()函数中，传进来的approve参数（也就是router），在达到条件后，会通过safeApprove()函数授权一个非常大的数值，如果该地址为恶意地址，则会造成不必要的损失，具体代码如下：

```
function approve(IERC20 erc, address approvee) internal {
    if (address(erc) != emptyAddr && erc.allowance(address(this), approvee) ==
0) {
        erc.safeApprove(approvee, uint256(- 1));
    }
}
```

- **安全建议**

对授权的地址进行严格检查，可将授权值相对减少。

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	零时科技内部审计工具包+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。

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。

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