智能合约安全审计报告

[2021]

目录

1 前言

慢雾安全团队于2021.03.29，收到BackFinance团队对BackFinance智能合约安全审计的申请，慢雾安全团队根据项目特点制定如下审计方案。

慢雾安全团队将采用“白盒为主，黑灰为辅”的策略，以最贴近真实攻击的方式，对项目进行安全审计。

慢雾科技项目测试方法：

测试方法	说明
黑盒测试	站在外部从攻击者角度进行安全测试。
灰盒测试	通过脚本工具对代码模块进行安全测试，观察内部运行状态，挖掘弱点。
白盒测试	基于项目的源代码，进行脆弱性分析和漏洞挖掘。

慢雾科技漏洞风险等级：

漏洞等级	说明
严重漏洞	严重漏洞会对项目的安全造成重大影响，强烈建议修复严重漏洞。
高危漏洞	高危漏洞会影响项目的正常运行，强烈建议修复高危漏洞。
中危漏洞	中危漏洞会影响项目的运行，建议修复中危漏洞。
低危漏洞	低危漏洞可能在特定场景中会影响项目的业务操作，建议项目方自行评估和考虑这些问题是否需要修复。
弱点	理论上存在安全隐患，但工程上极难复现。
增强建议	编码或架构存在更好的实践方法。

2 审计方法

慢雾安全团队智能合约安全审计流程包含两个步骤:

- 使用开源或内部自动化分析的工具对合约代码中常见的安全漏洞进行扫描和测试。
- 人工审计代码的安全问题，通过人工分析合约代码，发现代码中潜在的安全问题。

如下是合约代码审计过程中慢雾安全团队会重点审查的漏洞列表:

（其他未知的安全漏洞及审计项不包含在本次审计责任范围）

- 重入漏洞
- 重放漏洞
- 重排漏洞
- 短地址漏洞
- 拒绝服务漏洞
- 交易顺序依赖漏洞
- 条件竞争漏洞
- 权限控制漏洞
- 整数上溢/下溢漏洞
- 时间戳依赖漏洞
- 未声明的存储指针漏洞
- 算术精度误差漏洞
- tx.origin身份验证漏洞
- 假充值漏洞
- 变量覆盖漏洞
- Gas优化审计
- 恶意 Event 事件审计
- 冗余的回调函数
- 不安全的外部调用审计
- 函数状态变量可见性审计
- 业务逻辑缺陷审计
- 变量声明及作用域审计

3 项目概要

3.1 项目介绍

慢雾科技智能合约安全审计组：BackFinanace，

类型：DeFi

模块：Swap + Strategy + Loans + liquidation + ERC20 Token

代码行数：2822

复杂度：中等

工作量：10 个工作日，智能合约安全审计标准、相关说明及审计后证书查询：https://www.slowmist.com/service-smart-contract-security-audit.html

项目地址：

https://github.com/back-finance/back-finance-core

commit: 4829b4d926c6e77f9931e9a0db937e08409364ca

最后审计版本：

https://github.com/back-finance/back-finance-core

commit: 2a7fb5b9a7c2d60c24973074d2901c7cbf23b88b

请确认以上项目地址是否正确。

出具报告前需要项目方在区块浏览器开源验证，并给到相关合约地址。

3.2 漏洞信息

如下是本次审计发现的漏洞及漏洞的修复状态信息：

NO	标题	漏洞类型	漏洞等级	漏洞状态
N1	事件记录缺失	其它	建议	已修复
N2	emergencyWithdraw并没有emergency状态的判断	其它	建议	已忽略

NO	标题	漏洞类型	漏洞等级	漏洞状态
N3	swapOut可被恶意攻击者从合约中取走资金	业务逻辑缺陷审计	低	已忽略
N4	swapOut 函数没有严格的滑点检查存在三明治攻击	重排攻击	中	已确认
N6	poolList、pairList函数存在DoS	拒绝服务攻击	建议	已忽略
N7	权限过大：Developer 角色可以通过createPool 任意创建 pool	权限控制攻击	中	已确认
N8	权限过大：Developer角色可以通过createPair任意创建Pair	权限控制攻击	中	已确认
N9	addLiquidity需要添加滑点检查	业务逻辑缺陷审计	低	已忽略

4 审计详情

4.1 合约基础信息

如下是合约主网地址：

目前代码还未部署到主网。

4.2 函数可见性分析

在审计过程中，慢雾安全团队对核心合约的函数可见性进行分析，结果如下：

Configable			
Function Name	Visibility	Mutability	Modifiers
constructor	public	can modify state	-

Configable			
setupConfig	external	can modify state	onlyOwner

BXHStrategy			
Function Name	Visibility	Mutability	Modifiers
constructor	public	can modify state	-
setupConfig	external	can modify state	onlyOwner
initalize	external	can modify state	onlyDeveloper
harvest	external	can modify state	-
getRewardAmount	external	-	-
invest	external	can modify state	-
divest	external	can modify state	-
distributeReward	internal	can modify state	-
getPledgeAmount	external	-	-

BXHSwapper			
Function Name	Visibility	Mutability	Modifiers
constructor	public	can modify state	-
setupConfig	external	can modify state	onlyOwner
constructor	public	can modify state	-
sortTokens	internal	-	-
getPair	public	-	-

BXHSwapper			
getReserves	public	-	-
addLiquidity	external	can modify state	-
_getAmount	internal	-	-
removeLiquidity	external	can modify state	-
_addLiquidity	internal	can modify state	-
_swapForExactAmount	internal	can modify state	-
getLPAddress	external	-	-
getLiquidityAmount	public	-	-
swapOut	public	can modify state	-

BackConfig			
Function Name	Visibility	Mutability	Modifiers
constructor	public	can modify state	-
initialize	external	can modify state	-
changeDeveloper	external	can modify state	-
setWallets	external	can modify state	-
initParameter	external	can modify state	-
initPoolParams	external	can modify state	-
_setPoolValue	internal	can modify state	-
_setParams	internal	can modify state	-

BackConfig			
_setPoolParams	internal	can modify state	-
_setPrice	internal	can modify state	-
setTokenPrice	external	can modify state	-
setValue	external	can modify state	-
setPoolValue	external	can modify state	-
getValue	external	-	-
getPoolValue	external	-	-
setParams	external	can modify state	-
setPoolParams	external	can modify state	-
getParams	external	-	-
getPoolParams	external	-	-
calculateInterestRate	public	-	-
setInterestParams	public	can modify state	-
convertTokenAmount	external	-	-

BackPair			
Function Name	Visibility	Mutability	Modifiers
constructor	public	can modify state	-
setupConfig	external	can modify state	onlyOwner
initialize	external	can modify state	onlyPairFactory

BackPair			
setSwapper	external	can modify state	onlyDeveloper
setStrategy	external	can modify state	onlyDeveloper
invest	external	can modify state	onlyPlatform,updateInterest
divest	external	can modify state	onlyPlatform,updateInterest
repay	external	can modify state	onlyPlatform,updateInterest
reinvest	external	can modify state	onlyPlatform
claim	external	can modify state	onlyPlatform
getBorrowInfo	public	-	-
getTotalAssets	public	-	-
canLiquidation	public	-	-
liquidation	external	can modify state	onlyPlatform,updateInterest

BackPairFactory			
Function Name	Visibility	Mutability	Modifiers
constructor	public	can modify state	-
setupConfig	external	can modify state	onlyOwner
createPair	external	can modify state	onlyDeveloper
countPairs	external	-	-
getPairs	external	-	-

BackPlatform

BackPlatform			
Function Name	Visibility	Mutability	Modifiers
constructor	public	can modify state	-
setupConfig	external	can modify state	onlyOwner
receive	external	payable	-
deposit	external	payable	-
withdraw	external	can modify state	-
invest	external	payable	-
divest	external	can modify state	-
repay	external	payable	-
reinvest	external	can modify state	-
liquidation	external	payable	-
queryBack	external	-	-
mintBack	external	can modify state	-
claim	external	can modify state	-
_mintBack	internal	can modify state	-

ERC20Token			
Function Name	Visibility	Mutability	Modifiers
_mint	internal	can modify state	-
_burn	internal	can modify state	-

ERC20Token			
_transfer	private	can modify state	-
approve	external	can modify state	-
transfer	external	can modify state	-
transferFrom	external	can modify state	-

BackPool			
Function Name	Visibility	Mutability	Modifiers
_mint	internal	can modify state	-
_burn	internal	can modify state	-
_transfer	private	can modify state	-
approve	external	can modify state	-
transfer	external	can modify state	-
transferFrom	external	can modify state	-
constructor	public	can modify state	-
setupConfig	external	can modify state	onlyOwner
initialize	external	can modify state	onlyPoolFactory
sync	public	can modify state	_updateInterests
withdrawReserve	external	can modify state	onlyDeveloper
getTotalShare	public	-	-
claimBack	external	can modify state	onlyPlatform,_mintBack

BackPool			
queryBack	external	-	-
deposit	external	can modify state	onlyPlatform,_updateInterests,_mintBack,lock
withdraw	external	can modify state	onlyPlatform,_updateInterests,_mintBack,lock
borrow	external	can modify state	onlyPlatform,_updateInterests,_mintBack,lock
repay	external	can modify state	onlyPlatform,_updateInterests,_mintBack,lock

BackPoolFactory			
Function Name	Visibility	Mutability	Modifiers
constructor	public	can modify state	-
setupConfig	external	can modify state	onlyOwner
createPool	external	can modify state	onlyDeveloper
countPools	external	-	-
getPools	external	-	-

BackQuery			
Function Name	Visibility	Mutability	Modifiers
constructor	public	can modify state	-
tokenPrice	public	-	-
pairList	public	-	-
getUserInfo	public	-	-
poolList	public	-	-

BackQuery			
getBackInfo	public	-	-
tokenBasicInfo	public	-	-
batchGetConfigValues	external	-	-
getAssetInfo	external	-	-

BackToken			
Function Name	Visibility	Mutability	Modifiers
constructor	public	can modify state	-
enableMint	external	can modify state	-
setBoardRoom	external	can modify state	-
mintBoardRoom	external	can modify state	-
setWeights	external	can modify state	-
queryPool	external	-	-
mintPool	external	can modify state	-
mintWallet	external	can modify state	-
_update	internal	can modify state	-
_mint	internal	can modify state	-
_burn	internal	can modify state	-
_transfer	private	can modify state	-
approve	external	can modify state	-

BackToken			
transfer	external	can modify state	-
transferFrom	external	can modify state	-

MdexStrategy			
Function Name	Visibility	Mutability	Modifiers
constructor	public	can modify state	-
setupConfig	external	can modify state	onlyOwner
initalize	external	can modify state	onlyDeveloper
harvest	external	can modify state	-
getRewardAmount	external	-	-
invest	external	can modify state	-
divest	external	can modify state	-
distributeReward	internal	can modify state	-
getPledgeAmount	external	-	-

MdexSwapper			
Function Name	Visibility	Mutability	Modifiers
constructor	public	can modify state	-
setupConfig	external	can modify state	onlyOwner
constructor	public	can modify state	-
setSwapMining	external	can modify state	onlyDeveloper

MdexSwapper			
claimFee	external	can modify state	onlyDeveloper
getPair	public	-	-
getReserves	public	-	-
addLiquidity	external	can modify state	-
_getAmount	internal	-	-
removeLiquidity	external	can modify state	-
_addLiquidity	internal	can modify state	-
_swapForExactAmount	internal	can modify state	-
getLPAddress	external	-	-
getLiquidityAmount	public	-	-
getAmountOut	external	-	-
getAmountIn	external	-	-
swapOut	public	can modify state	-

4.3 漏洞详情

[N1] [建议] 事件记录缺失

漏洞类型: 其它

详细内容

contracts/BackConfig.sol 中多个方法如：changeDeveloper，setWallets，setInterestParams等方法建议添加事件进行记录，便于链下对操作的监控，以及便于社区用户的审查。

```
function changeDeveloper(address _developer) external {
    require(msg.sender == owner || msg.sender == developer, "Back: Config
FORBIDDEN");
    developer = _developer;
}
```

```
function setWallets(bytes32[] calldata _names, address[] calldata _wallets) external
{
    require(msg.sender == owner || msg.sender == developer, "Back: ONLY DEVELOPER");
    require(_names.length == _wallets.length ,"Back: WALLETS LENGTH MISMATCH");
    for(uint i = 0; i < _names.length; i ++)
    {
        wallets[_names[i]] = _wallets[i];
    }
}
```

```
function setInterestParams(address pool, uint[] memory params1, uint[] memory
params2, uint[] memory params3) public {
    require(msg.sender == owner || msg.sender == governor || msg.sender == developer,
"Back: FORBIDDEN");

    if(params1.length == 4) {
        interestParams[pool][0] = InterstStruct(params1[0], params1[1], params1[2],
params1[3]);
    }

    if(params2.length == 4) {
```

```
        interestParams[pool][1] = InterstStruct(params2[0], params2[1], params2[2],
params2[3]);
    }

    if(params3.length == 4) {
        interestParams[pool][2] = InterstStruct(params3[0], params3[1], params3[2],
params3[3]);
    }
}
```

其它相关缺失事件记录的敏感方法也建议添加事件记录。

解决方案

对相关方法添加Log。

漏洞状态

已修复

[N2] [建议] emergencyWithdraw并没有emergency状态的判断

漏洞类型: 其它

详细内容

contracts/bxh/airdroppool.sol 208-217行,

```
// Withdraw without caring about rewards. EMERGENCY ONLY.
function emergencyWithdraw(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    uint256 amount = user.amount;
    user.amount = 0;
    user.rewardDebt = 0;
```

```
        pool.lpToken.safeTransfer(address(msg.sender), amount);
        emit EmergencyWithdraw(msg.sender, _pid, amount);
    }
```

contracts/bxh/bxhpool.sol 487-495行，

```
    // Withdraw without caring about rewards. EMERGENCY ONLY.
    function emergencyWithdraw(uint256 _pid) public notPause {
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            emergencyWithdrawBXHAndToken(_pid, msg.sender);
        } else {
            emergencyWithdrawBXH(_pid, msg.sender);
        }
    }
```

两个方法看注释及方法的预期作用，是在紧急状态下使用的，但并没有emergency的判断

解决方案

建议增加emergency状态判断。

漏洞状态

已忽略；无安全影响

[N3] [低] swapOut可被恶意攻击者从合约中取走资金

漏洞类型: 业务逻辑缺陷审计

详细内容

swapOut 函数可以任意兑换 token, 因为fee是在 contracts/MdexSwapper.sol 合约中，如果在mdex上构造一个 mdx token + 恶意的交易对，就能通过 swapOut 将 fee 薅走。

contracts/MdexSwapper.sol 356-365行，

```
function swapOut(address fromToken, address toToken, uint amountIn, address to, uint
deadline) public {
    require(mdexFactory.getPair(fromToken, toToken) != address(0), "NOT SUPPORT
NOW");
    if(amountIn > 0) {
        address[] memory path = new address[](2);
        path[0] = fromToken;
        path[1] = toToken;
        TransferHelper.safeApprove(fromToken, address(mdexRouter), amountIn);
        mdexRouter.swapExactTokensForTokens(amountIn, 1, path, to, deadline);
    }
}
```

contracts/BXHSwapper.sol 224-233行,

```
function swapOut(address fromToken, address toToken, uint amountIn, address to, uint
deadline) public {
    require(bxhFactory.getPair(fromToken, toToken) != address(0), "NOT SUPPORT NOW");
    if(amountIn > 0) {
        address[] memory path = new address[](2);
        path[0] = fromToken;
        path[1] = toToken;
        TransferHelper.safeApprove(fromToken, address(bxhRouter), amountIn);
        bxhRouter.swapExactTokensForTokens(amountIn, 1, path, to, deadline);
    }
}
```

mdexFactory及bxhFactory 的合约地址是外部的，可以添加交易对则可被攻击成功。

解决方案

严格控制交易对，不依赖于外部合约

漏洞状态

已忽略；业务逻辑上不会存钱到合约中，唯一的可能是用户转钱进来。

[N4] [中] swapOut 函数没有严格的滑点检查存在三明治攻击

漏洞类型: 重排攻击

详细内容

contracts/MdexSwapper.sol 356-365行，

```
function swapOut(address fromToken, address toToken, uint amountIn, address to, uint
deadline) public {
    require(mdexFactory.getPair(fromToken, toToken) != address(0), "NOT SUPPORT
NOW");
    if(amountIn > 0) {
        address[] memory path = new address[](2);
        path[0] = fromToken;
        path[1] = toToken;
        TransferHelper.safeApprove(fromToken, address(mdexRouter), amountIn);
        mdexRouter.swapExactTokensForTokens(amountIn, 1, path, to, deadline);
    }
}
```

contracts/BXHSwapper.sol 224-233行，

```
function swapOut(address fromToken, address toToken, uint amountIn, address to, uint
deadline) public {
    require(bxhFactory.getPair(fromToken, toToken) != address(0), "NOT SUPPORT NOW");
```

```
        if(amountIn > 0) {
            address[] memory path = new address[](2);
            path[0] = fromToken;
            path[1] = toToken;
            TransferHelper.safeApprove(fromToken, address(bxhRouter), amountIn);
            bxhRouter.swapExactTokensForTokens(amountIn, 1, path, to, deadline);
        }
    }
```

参考：https://medium.com/coinmonks/demystify-the-dark-forest-on-ethereum-sandwich-attacks-5a3aec9fa33e

解决方案

建议添加滑点检查，确保swap的时候滑点是在预期内的。

漏洞状态

已确认；可以增加权限限制来解决

[N6] [建议] poolList、pairList函数存在DoS

漏洞类型: 拒绝服务攻击

详细内容

contracts/BackQuery.sol 479-499行，

```
function poolList() public view returns (PoolInfo[] memory pools) {
    address[] memory tokens = IBackPoolFactory(backPoolFactory).getPools();
    pools = new PoolInfo[](tokens.length);
    for (uint256 i = 0; i < tokens.length; i++) {
        address _supplyToken = IBackPool(tokens[i]).supplyToken();
        uint _totalBorrow = IBackPool(tokens[i]).totalBorrow();
        uint _totalShare = IBackPool(tokens[i]).getTotalShare();
        uint _totalSupply = IBackPool(tokens[i]).totalSupply();
        uint _backPerDeposit = IBackPool(tokens[i]).backPerDeposit();
```

```
        uint _backPerBorrow = IBackPool(tokens[i]).backPerBorrow();
        uint _interestPerBorrow = IBackPool(tokens[i]).interestPerBorrow();
        uint _interestRate = IBackPool(tokens[i]).interestRate();
        uint _price = IBackConfig(config).prices(_supplyToken);
        // (uint _backWeight,, ) = IBackToken(backToken).deposits(tokens[i]);
        uint _backWeight = 0;
        uint _balance = IBackPool(tokens[i]).balanceOf(msg.sender);
        uint _depositPercent = IBackConfig(config).getPoolValue(tokens[i],
bytes32('POOL_BACK_DEPOSIT_PERCENT'));
        pools[i] = PoolInfo(tokens[i], _supplyToken, _totalShare, _totalBorrow,
_totalSupply, _backPerDeposit,
         _backPerBorrow, _interestPerBorrow, _interestRate, _price, _backWeight,
_depositPercent, _balance);
    }
}
```

contracts/BackQuery.sol 434-455行,

```
function pairList() public view returns (PairInfo[] memory pairs) {
    uint256 count = IBackPairFactory(pairFactory).countPairs();
    pairs = new PairInfo[](count);
    for (uint256 i = 0; i < count; i++) {
        address _pair = IBackPairFactory(pairFactory).allPairs(i);
        address _token0 = IBackPair(_pair).token0();
        address _token1 = IBackPair(_pair).token1();
        uint256 _pid =
            IBackPairFactory(pairFactory).pidOfPair(_token0, _token1);
        uint _totalPledge = IStrategy(IBackPair(_pair).strategy()).totalPledge();
```

```
        address _swapper = IBackPair(_pair).swapper();
        (uint _reserve0, uint _reserve1) = ISwapper(_swapper).getReserves(_token0,
_token1);

        uint _borrow0 = IBackPair(_pair).totalBorrowToken0();
        uint _borrow1 = IBackPair(_pair).totalBorrowToken1();
        uint _leverageRate = IBackConfig(config).getPoolValue(_pair,
bytes32('PAIR_LEVERAGE_RATE'));
        uint _liquidationRate = IBackConfig(config).getPoolValue(_pair,
bytes32('PAIR_LIQUIDATION_RATE'));

        pairs[i] = PairInfo(_pair, _token0, _token1, _pid, _totalPledge, _reserve0,
_reserve1,
         _borrow0, _borrow1, ISwapperPair(ISwapper(_swapper).getPair(_token0,
_token1)).totalSupply(), _leverageRate, _liquidationRate);
    }
}
```

函数的 for 循环没有分批处理的机制，当 length 较大的时候会有 DoS (Out Of Gas) 的问题。

解决方案

建议加上分批处理的机制。

漏洞状态

已忽略；项目方明确不会创建这么多池子

[N7] [中] 权限过大：Developer 角色可以通过createPool 任意创建 pool

漏洞类型: 权限控制攻击

详细内容

Developer 角色可以通过 createPool 任意创建 pool, 存在权限过大和创建恶意的 pool 的问题。

contracts/BackPoolFactory.sol 14-31行,

```
function createPool(address _supplyToken) onlyDeveloper external returns (address
pool) {




    require(getPool[_supplyToken] == address(0), "ALREADY CREATED");

    // create pol
    bytes32 salt = keccak256(abi.encodePacked(_supplyToken));
    bytes memory bytecode = type(BackPool).creationCode;
    assembly {
        pool := create2(0, add(bytecode, 32), mload(bytecode), salt)
    }
    getPool[_supplyToken] = pool;

    allPools.push(pool);
    isPool[pool] = true;
    IConfig(config).initPoolParams(pool);
    BackPool(pool).setupConfig(address(config));
    BackPool(pool).initialize(_supplyToken);
    emit PoolCreated(_supplyToken, pool);
}
```

解决方案

建议需要维护 supplyToken 的白名单列表，且 Developer 角色需要将权限转移给 timelock 或治理合约。

漏洞状态

已确认；权限之后会设计交给governance

[N8] [中] 权限过大：Developer角色可以通过createPair任意创建Pair

漏洞类型: 权限控制攻击

详细内容

Developer 角色可以通过 createPair 任意创建 Pair, 存在权限过大和创建恶意的 Pair 的问题。

contracts/BackPairFactory.sol 16-40行,

```
function createPair(address token0, address token1) onlyDeveloper external returns
(address pair)  {
    // require(getPair[token0][token1] == address(0), "ALREADY CREATED");
    require(token0 != token1, "SAME TOKEN");
    uint pid = cntOfPair[token0][token1];

    // create pool
    bytes32 salt = keccak256(abi.encodePacked(token0, token1, pid));
    bytes memory bytecode = type(BackPair).creationCode;
    assembly {
        pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
    }
    getPair[token0][token1][pid] = pair;
    getPair[token1][token0][pid] = pair;
    pidOfPair[token0][token1] = pid;
    pidOfPair[token1][token0] = pid;
    cntOfPair[token0][token1]++;
    cntOfPair[token1][token0]++;

    allPairs.push(pair);
    isPair[pair] = true;
    IConfig(config).initPoolParams(pair);
```

```
    BackPair(pair).setupConfig(address(config));
    BackPair(pair).initialize(token0, token1);
    emit PairCreated(token0, token1, pair);
}
```

解决方案

Developer 角色需要将权限转移给 timelock 或治理合约。

漏洞状态

已确认；权限之后会设计交给governance

[N9] [低] addLiquidity需要添加滑点检查

漏洞类型: 业务逻辑缺陷审计

详细内容

contracts/BXHSwapper.sol 94-108,

```
function addLiquidity(address token0, address token1, uint deadline, address to)
external returns (uint liquidity){
uint balance0 = IERC20(token0).balanceOf(address(this));
uint balance1 = IERC20(token1).balanceOf(address(this));

(uint reserve0, uint reserve1) = getReserves(token0, token1);
if(balance0 * reserve1 >= balance1 * reserve0) {
    uint amountIn = _getAmount(balance0, balance1, reserve0, reserve1);
    exeSwapOut(token0, token1, amountIn, address(this), deadline);
} else {
    uint amountIn = _getAmount(balance1, balance0, reserve1, reserve0);
    exeSwapOut(token1, token0, amountIn, address(this), deadline);
}
```

```
liquidity = _addLiquidity(token0, token1, deadline, to);
}
```

及 contracts/MdexSwapper.sol 216-230,

```
function addLiquidity(address token0, address token1, uint deadline, address to)
external returns (uint liquidity) {
    uint balance0 = IERC20(token0).balanceOf(address(this));
    uint balance1 = IERC20(token1).balanceOf(address(this));

    (uint reserve0, uint reserve1) = mdexFactory.getReserves(token0, token1);
    if(balance0 * reserve1 >= balance1 * reserve0) {
        uint amountIn = _getAmount(balance0, balance1, reserve0, reserve1);
        exeSwapOut(token0, token1, amountIn, address(this), deadline);
    } else {
        uint amountIn = _getAmount(balance1, balance0, reserve1, reserve0);
        exeSwapOut(token1, token0, amountIn, address(this), deadline);
    }

    liquidity = _addLiquidity(token0, token1, deadline, to);
}
```

相关的addLiquidity并没有进行相关的滑点检查，导致用户资金可能存在被套利机器人套走的风险。

解决方案

addLiquidity添加对应的滑点检查, 解决方案参考 https://github.com/Uniswap/uniswap-v2-periphery/blob/master/contracts/UniswapV2Router02.sol#L61

漏洞状态

已忽略；项目方认为此处滑点造成的资产损失在可承受的范围内。

5 审计结果

审计编号	审计团队	审计日期	审计结果
0X002104150001	SlowMist Security Team	2021.03.29 - 2021.04.12	中风险

总结：

慢雾安全团队采用人工结合内部工具对代码进行分析，审计期间发现了 3 个中危漏洞， 2 个低危漏洞， 3 个增强建议。其中 3 个中危漏洞已确认修复中； 2 个低危漏洞， 2 个增强建议暂时被忽略；其它所有漏洞均已修复。 目前代码还未部署到主网。

6 声明

厦门慢雾科技有限公司(下文简称 ”慢雾”) 仅就本报告出具前项目方已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后项目方发生或存在的未知漏洞及安全事件，慢雾无法判断其安全状况，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慢雾提供的文件和资料(简称“已提供资料”)。慢雾假设: 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，慢雾对由此而导致的损失和不利影响不承担任何责任，慢雾仅对该项目的安全情况进行约定内的安全审计并出具了本报告，慢雾不对该项目背景及其他情况进行负责。

官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号