Smart contract security audit report

Hexch smart contract security audit report

Audit Team： Noneage security team

Audit date： April 8, 2021

Hexch Smart Contract Security Audit Report

1. Overview

On April 7, 2021, the security team of LS Technology received the security audit request of the **Hexch project**. The team will conduct a report on the **Hexch smart contract** from April 7, 2021 to April 8, 2021. During the audit process, the security audit experts of Zero Hour Technology communicate with the relevant interface people of the Hexch project, maintain information symmetry, conduct security audits under controllable operational risks, and try to avoid project generation and operation during the test process. Cause risks.

Through communicat and feedback with Hexch project party, it is confirmed that the loopholes and risks found in the audit process have been repaired or within the acceptable range. The result of this Hexch smart contract security audit: **passed.**

Audit Report MD5：31CE5ECD04016D54789A707B2BAD9C32

2. Background

2.1 Project Description

> Project name: Hexch
>
> official website: https://hexch.io
>
> Contract type: DeFi Token contract
>
> Code language: Solidity
>
> Deploy public chain: Huobi ECO Chain
>
> Contract address: 0xe3ffa0ca53f8729daf54b866bd34a6c132e657a3

2.2 Audit Range

Hexch official chain contract：https://hecoinfo.com/address/0xe3ffa0ca53f8729daf54b866bd34a6c132e657a3#code

2.3 Security Audit List

The security experts of Noneage Technology conduct security audits on the security audit list within the agreement, The scope of this smart contract security audit does not include new attack methods that may appear in the future, does not include the code after contract upgrades or tampering, and is not included in the subsequent cross-country, does not include cross-chain

deployment, does not include project front-end code security and project platform server security.

This smart contract security audit list includes the following:

- Integer overflow
- Reentry attack
- Floating point numbers and numerical precision
- Default visibility
- Tx.origin authentication
- Wrong constructor
- Return value not verified
- Insecure random numbers
- Timestamp dependency
- Transaction order is dependent
- Delegatecall
- Call
- Denial of service
- Logic design flaws
- Fake recharge vulnerability
- Short address attack
- Uninitialized storage pointer
- Additional token issuance
- Frozen account bypass
- Access control
- Gas usage

3. Contract Structure Analysis

3.1 Directory Structure

└─hexch.sol

3.2 AuctionPool contract

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)

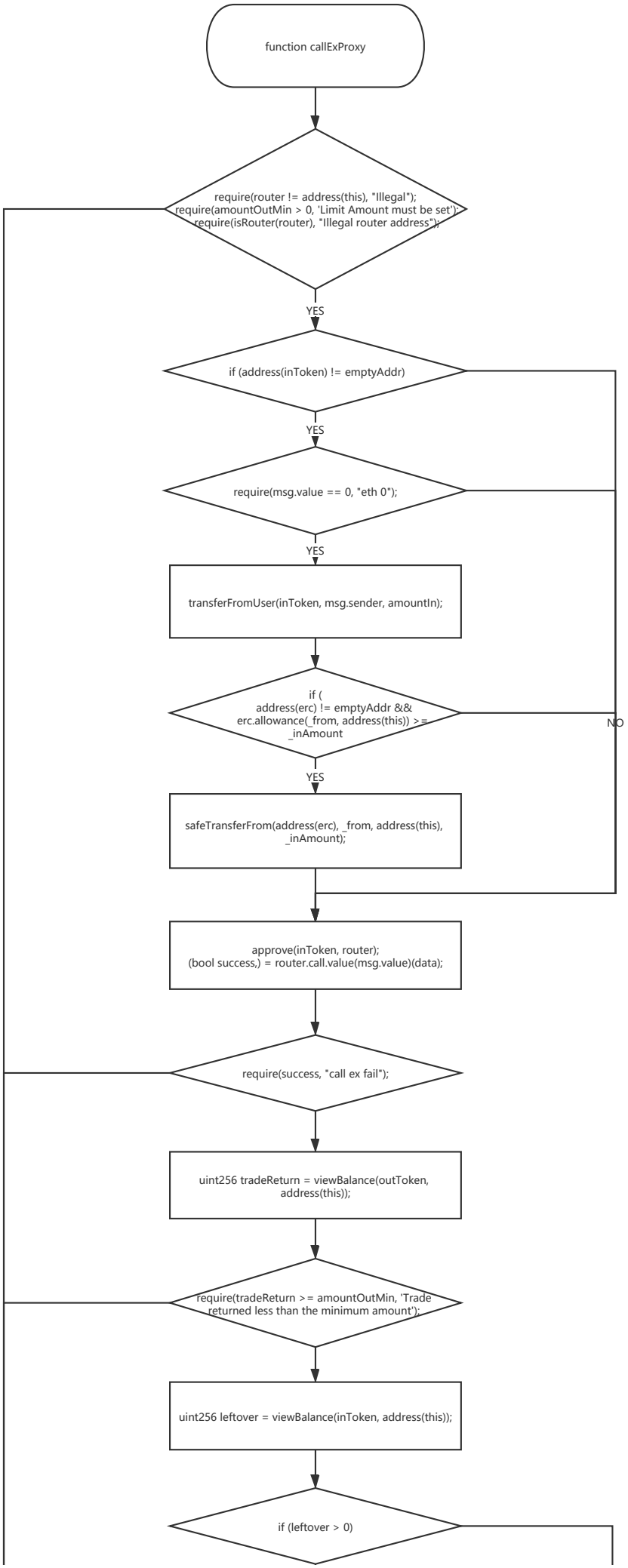
SafeMath

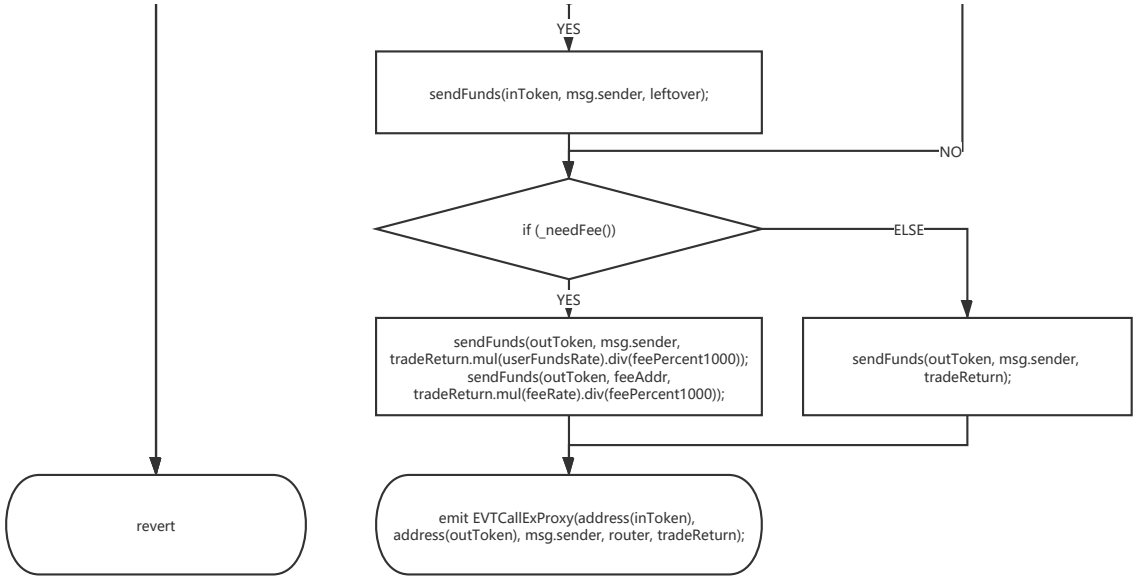
- add(uint256 a, uint256 b)
- sub(uint256 a, uint256 b)
- mul(uint256 a, uint256 b)
- div(uint256 a, uint256 b)
- mod(uint256 a, uint256 b)

3.5 Contract Logic Flow Chart

Through the security audit of the **Hexch contract**, the security auditor listed the code flow chart of part of the contract logic in the audit process, as follows:

Part of swapV1 contract logic: 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)

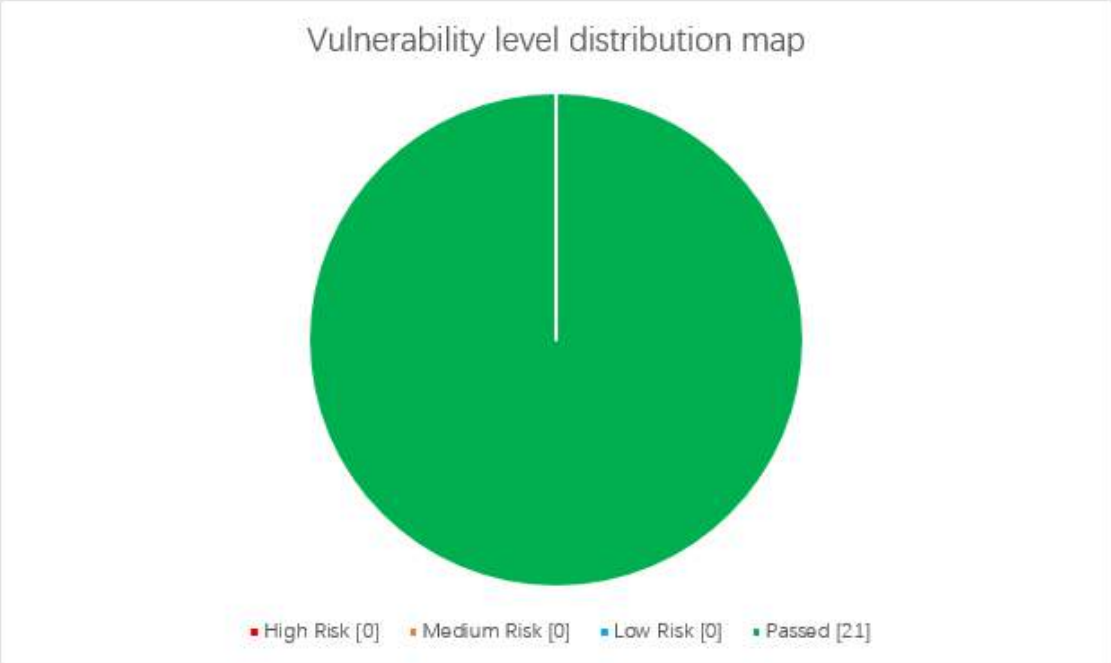


4. Audit Details

4.1 Vulnerabilities Distribution

Vulnerabilities in this security audit are distributed by risk level, as follows:

Vulnerability level distribution			
High risk	Medium risk	Low risk	Passed
0	0	0	21



This smart contract security audit has 0 high-risk vulnerabilities, 0 medium-risk vulnerabilities, 0 low-risk vulnerabilities, and 21 passed, with a high security level.

4.2 Vulnerabilities Details

A security audit was conducted on the smart contract within the agreement, and no security vulnerabilities that could be directly exploited and generated security problems were found, and the security audit was passed.

4.3 Other Risks

Other risks refer to the code that smart contract security auditors consider to be risky. Under certain circumstances, it may affect the stability of the project, but it cannot constitute a security issue that directly harms.

4.3.1 Excessive admin rights

- **Issue causes**

The administrator authority in the smart contract is too large. When the administrator key is lost or controlled by a malicious person, it will affect the normal operation of the project.

- **Question detail**

Through the audit contract, it is found that there are many settings in the swapV1 contract. The transfer and self-destruction are directly or indirectly controlled by the administrator. If the administrator key is lost or controlled by a malicious person, the project will not operate normally or the contract will be destroyed. The specific code is as follows:

```
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);
```

```
}
```

- **Safety advice**

The time lock can be used to prevent a large number of tokens from being transferred; the private key of the deployer's address needs to be stored safely and effectively to avoid loss or acquisition by malicious persons.

4.3.2 Over-authorization problem

- **Issue causes**

When the administrator authorizes a parameter or address, the value is given greater authority. If the assigned address performs dangerous operations on the contract, it will affect the stability of the project.

- **Question detail**

Through the audit contract, it is found that there is an over-authorization problem in the swapV1 contract. For example, in the approve() function, the incoming approve parameter (that is, the router) will be authorized by the safeApprove() function to a very large value after the condition is met. If the address is malicious, it will cause unnecessary losses. The specific code is as follows:

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

- **Safety advice**

Strictly checking the authorized address can reduce the authorized value relatively.

5. Security Audit Tool

Tool name	Tool Features
Oyente	Can be used to detect common bugs in smart contracts
securify	Common types of smart contracts that can be verified
MAIAN	Multiple smart contract vulnerabilities can be found and classified
Noneage Internal Toolkit	Noneage Technology Internal Security Audit Toolkit + https://audit.noneage.com

6. Vulnerability assessment criteria

Vulnerability level	Vulnerability description
High risk	Vulnerabilities that can directly lead to the loss of contracts or users' digital assets, such as integer overflow vulnerabilities, false recharge vulnerabilities, re-entry vulnerabilities, illegal token issuance, etc. Vulnerabilities that can directly cause the ownership change of the token contract or verification bypass, such as: permission verification bypass, call code injection, variable coverage, unverified return value, etc. Vulnerabilities that can directly cause the token to work normally, such as denial of service vulnerabilities, insecure random numbers, etc.
Medium risk	Vulnerabilities that require certain conditions to trigger, such as vulnerabilities triggered by the token owner's high authority, and transaction sequence dependent vulnerabilities. Vulnerabilities that cannot directly cause asset loss, such as function default visibility errors, logic design flaws, etc.
Low risk	Vulnerabilities that are difficult to trigger, or vulnerabilities that cannot lead to asset loss, such as vulnerabilities that need to be triggered at a cost higher than the benefit of the attack, cannot lead to incorrect coding of security vulnerabilities.

零时科技
NONEAGE

Telephone：86-17391945345 18511993344
Email ：support@noneage.com
Site ：www.noneage.com
Weibo ：weibo.com/noneage