# Beosin's Analysis of Team Finance's \$13M Exploit



On October 27, 2022, Beosin EagleEye reported that Team Finance on Ethereum was exploited for over \$13M. The hacker illegally migrated WTH, CAW, USDC, TSUKA tokens from V2 liquidity pool to V3 liquidity pool by exploiting the vulnerability of the migrate function in LockToken contract, disrupted the Initialize price of V3 liquidity pool via sqrtPriceX96 to obtain a large amount of refund arbitrage. Beosin security team analyzed the incident and the results are as follows.

## Related information

## Attack transaction

oxb2e3ea72d353da43a2ac9a8f167ofd16463ab370e563b9b5b26119b2601277ce

#### Attacker's address

0x161cebB807Ac181d5303A4cCec2FC580CC5899Fd

oxBa399a258o785A2dEd74oF5e3oEC89Fb3E617e6E

### **Attack contract**

oxCFFo7C4e6aa9E2fEco4DAaF5f41d1b1of3adAdF4

#### Victim contract

## oxE2fE53oCo47f2d85298bo7D9333Co5737f1435fB

## **Attack Flow**

1. The migrate function of the victim contract (LockToken contract) supports the user to migrate the specified Uniswap-V2 liquidity to Uniswap-V3 liquidity. A part of the tokens will be refunded to the user based on the price after migration. The call to the function require: lock ID, lock time, and withdrawable address.

```
function migrate(
    uint256 _id,
    IV3Migrator.MigrateParams calldata params,
    bool noLiquidity,
    uint160 sqrtPriceX96,
    bool _mintNFT

1537    )
    issa external
    payable
    whenNotPaused
    nonReentrant

1542    {
        require(address(nonfungiblePositionManager) != address(0), "NFT manager not set");
        require(address(v3Migrator) != address(0), "v3 migrator not set");
        require(address(v3Migrator) != address(0), "v3 migrator not set");
        require(block.timestamp < lockedERC20.unlockTime, "Unlock time already reached");
        require(_msgSender() == lockedERC20.withdrawalAddress, "Unauthorised sender");
        require(!lockedERC20.withdrawalAddress, "Unauthorised sender");
        require(!lockedERC20.withdrawalAddress, "Unauthorised sender");
        require(!lockedERC20.withdrawalAddress, "Unauthorised sender");
```

```
uint256 refundEth = address(this).balance - ethBalanceBefore;
(bool refundSuccess,) = _msgSender().call.value(refundEth)("");
require(refundSuccess, 'Refund ETH failed');

uint256 token0BalanceAfter = IERC20(params.token0).balanceOf(address(this));
uint256 refundToken0 = token0BalanceAfter - token0BalanceBefore;

if( refundToken0 > 0 ) {
    require(IERC20(params.token0).transfer(_msgSender(), refundToken0));
}

uint256 token1BalanceAfter = IERC20(params.token1).balanceOf(address(this));
uint256 refundToken1 = token1BalanceAfter - token1BalanceBefore;

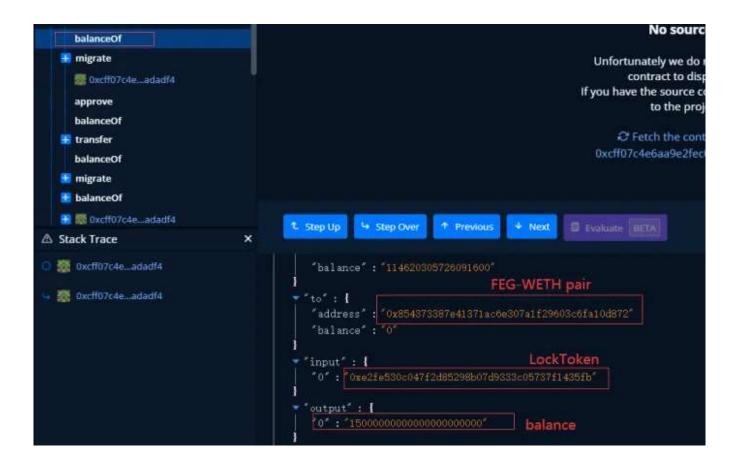
if( refundToken1 > 0 ) {
    require(IERC20(params.token1).transfer(_msgSender(), refundToken1));
}
```

- 2. Pre-attack preparation: the attacker 0x161ce...5899Fd first deployed the attack contract 0xCFF07C4e6aa9E2fEc04DAaF5f41d1b10f3adAdF4 as well as created the token contract 0x2d4abfdcd1385951df4317f9f3463fb11b9a31df (token A).
- 3. The attacker calls the lockToken function in the LockToken contract, performs four locks of token A created by himself, and sets the withdrawal address to the attack contract address, and obtains four NFTs as LP (ids 15324, 15325, 15326, 15327).

The lockToken function can lock the user's tokens and mint a NFT token as a LP token. The type, number, withdraw address, and the locking time of the locked token can all be specified.

4. Call the extendLockDuration function in the LockToken contract to adjust the locking time corresponding to each NFT token. At this point, the preparation is completed.

5. The attack contract queries the LockToken contract for the number of specified LP tokens and returns the result as part of the attack parameters.



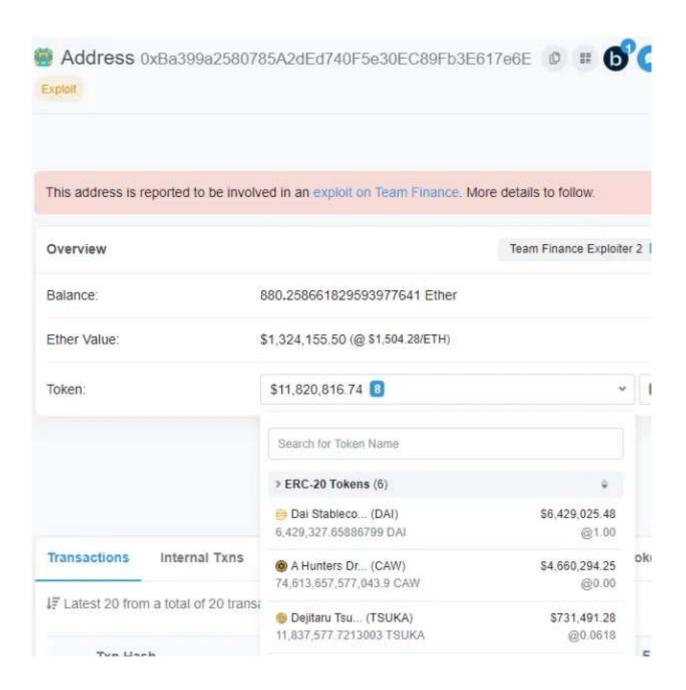
The attacker calls the migrate function, and due to the prior preparation of the NFT, the verification of the migrate function regarding the id and lock time, and the withdraw address are all bypassed. The NFT id obtained by locking the token A can participate in the migration of the FEG-WETH pair, without detecting whether the user's lock is the

same as the one being operated, and the parameter sqrtPriceX96, which is related to the price calculation of the UNI-V3 migration, is also input by the user.

```
″_id″:″15324″
     "params" : {
       amountOMin": "0"
      "amount1Min": "0"
      "deadline": "1666859863"
      "fee": "500"
      "liquidityToMigrate": "150000000000000000000000" 🗟
                                                                FEG-WETH pair
      "pair": "0x854373387e41371ac6e307a1f29603c6fa10d872"
       percentageToMigrate":1
      "recipient": "Oxba399a2580785a2ded740f5e30ec89fb3e617e6e"
      "refundAsETH" : true
      "tickLower": "-100"
      "tickUpper": "100"
                                                                   token A
      "token0": "0x2d4abfdcd1385951df4317f9f3463fb11b9a31df"
       token1": "0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2"
                                                                 WETH
  "noLiquidity": true
  "sqrtPriceX96": "79210883607084793911461085816"
   _mintNFT": false
[OUTPUT] ": "Ox"
  gas": {
```

```
function migrate(
  uint256 id,
   IV3Migrator.MigrateParams calldata params,
   bool noLiquidity,
                                                     Controllable parameters
  uint160 sqrtPriceX96,
   bool _mintNFT
payable
whenNotPaused
nonReentrant
   require(address(nonfungiblePositionManager) != address(θ), "NFT manager not set");
   require(address(v3Migrator) != address(0), "v3 migrator not set");
   Items memory lockedERC20 = lockedToken[_id];
   require(block.timestamp < lockedERC20.unlockTime, "Unlock time already reached");
   require(_msgSender() == lockedERC20.withdrawalAddress, "Unauthorised sender");
   require(!lockedERC20.withdrawn, "Already withdrawn");
   uint256 totalSupplyBeforeMigrate = nonfungiblePositionManager.totalSupply();
    //scope for solving stack too deep error
        uint256 ethBalanceBefore = address(this).balance;
        uint256 token@BalanceBefore = IERC2@(params.token@).balanceOf(address(this));
       uint256 token1BalanceBefore = IERC20(params.token1).balanceOf(address(this));
        if(noLiquidity) {
            v3Migrator.createAndInitializePoolIfNecessary(params.token0, params.token1, params.fee, sqrtPriceX96);
        IERC20(params.pair).approve(address(v3Migrator), params.liquidityToMigrate);
        v3Migrator.migrate(params);
```

6. The attack contract uses four NFTs prepared in advance to obtain the migration refunds for the four tokens: WETH, DAI, CAW, and TSUKA, all of which were sent to the 0xBa399a2580785A2dEd740F5e30EC89Fb3E617e6E address.



# **Vulnerability analysis**

This attack mainly exploits a vulnerability in the migrate function of the LockToken contract. The validation of migrate is easily bypassed and can manipulate the price during migration.

## **Fund flow**

The stolen funds are 880.258 ETH, 642,9327.6 DAI, 74,6136,5757,7043 CAW, 1183,7577.7 TSUKA, with a total value of about \$13 million, which remain in the attacker's address oxBa399a258o785A2dEd74oF5e3oEC89Fb3E617e6E.

## **Summary**

In response to this incident, Beosin security team recommends that

- 1. Carefully verify parameters of important functions, especially user-controllable parameters.
- 2. Choose a professional security audit company before the project goes live.

# Contact

If you have need any blockchain security services, please contact us:

Website Email Official Twitter Alert Telegram LinkedIn