**The Main Cause of Vee Finance Attack**



**Background**

On September 21, 2021, Vee Finacne was attacked and contract funds were stolen.

**Project Description**

Vee Finance is a lending protocol that is mainly forked from Compound Protocol and add leveraged trading logic on this basis. Users can obtain loan vouchers through mortgage assets, and the loan vouchers can be leveraged in the protocol（用户可以通过抵押资产获得贷款凭证，贷款凭证可以在协议中杠杆化。）. When performing leveraged transactions, users will lend funds from the contract, and then create an order through the VeeProxyController contract. When the order is created, the contract will swap the borrowed funds into the target token in Pangolin. When the order expires or the stop-profit and stop-loss price is reached, reverse swap will be performed, and then the loan will be returned.

**Main Cause**

The main cause of the accident was that in the process of creating an order for leveraged trading, only the price of the Pangolin pool was used by the oracle as the source of price feed, and the pool price fluctuated more than 3%. The oracle refreshed the price, causing the attacker to manipulate the price of the Pangolin pool. Manipulating the price of the Vee Finance oracle machine and the acquisition of the oracle machine price were not processed for decimals, resulting in the expected slippage check before the swap did not work.

**Direct Cause**

1. The oracle machine has a single source of price feed, and the refresh conditions are affected by the real-time number of tokens in the Pangolin pool (the pool price fluctuates by 3%, and it will be refreshed).

2. Price acquisition has not been processed for decimals(价格获取没有处理小数).

**Indirect Cause**

1. The check on the contract call is bypassed.

2. The target pair of margin trading is not whitelisted.

**Detailed Analysis**

1. When performing margin trading, the createOrderERC20ToERC20 function in the following code block will be called to create an order.

2. When an order is created, the token exchange will be carried out through line 5 of the following code block.

3. Before the token exchange, the expected slippage will be checked through the getAmountOutMin function on line 9 of the following code block.

4. During slippage check, the priceA and PriceB quotes of the oracle will be obtained through lines 12 and 13 of the following code block, and then the number of TokenA that can be exchanged for TokenB at the current price is calculated through line 15 of the following code block. . Finally, compare with the number of tokens acquired in the Pangolin pool. If the number of TokenB tokens that can be exchanged in the pool is greater than or equal to the expected number of TokenB that can be exchanged using the oracle, then it can be judged that the pool price is correct and not controlled, and the order creation logic is continued.

***function createOrderERC20ToERC20****(address orderOwner,CreateParams memory createParams)****external veeLock****(uint8(VeeLockState.LOCK\_CREATE))****payable returns****(bytes32 orderId){*

*address tokenA = CErc20Interface(createParams.ctokenA).underlying();*

*address tokenB = CErc20Interface(createParams.ctokenB).underlying();*

*commonCheck(orderOwner,createParams.stopHighPairPrice,createParams.stopLowPairPrice,createParams.amountA,createParams.expiryDate,createParams.leverage,createParams.ctokenA,tokenA,tokenB);*

*uint256[] memory amounts = swapERC20ToERC20(tokenA,tokenB,calcSwapAmount(createParams.amountA,createParams.leverage),getAmountOutMin(createParams));*

*orderId = onOrderCreate(orderOwner,createParams,amounts,tokenA,tokenB);*

*}*

***function getAmountOutMin****(CreateParams memory createParams)****internal returns****(uint256 amountOutMin) {*

*address tokenA = getUnderlying(createParams.ctokenA);*

*address tokenB = getUnderlying(createParams.ctokenB);*

*uint256 priceA = IPriceOracle(oracle).getUnderlyingPrice(createParams.ctokenA);*

*uint256 priceB = IPriceOracle(oracle).getUnderlyingPrice(createParams.ctokenB);*

*uint256 swapAmountA = calcSwapAmount(createParams.amountA,createParams.leverage);*

*uint256 amountFromOracle = priceA \* swapAmountA / priceB;*

*按照当前价格计算swapAmountA相当于多少amountB*

*uint256 amountOut = getAmountOut(tokenA,tokenB,swapAmountA);*

*swapAforB:计算能够得到多少amountB*

*amountOutMin = amountFromOracle \* 95 / 100;*

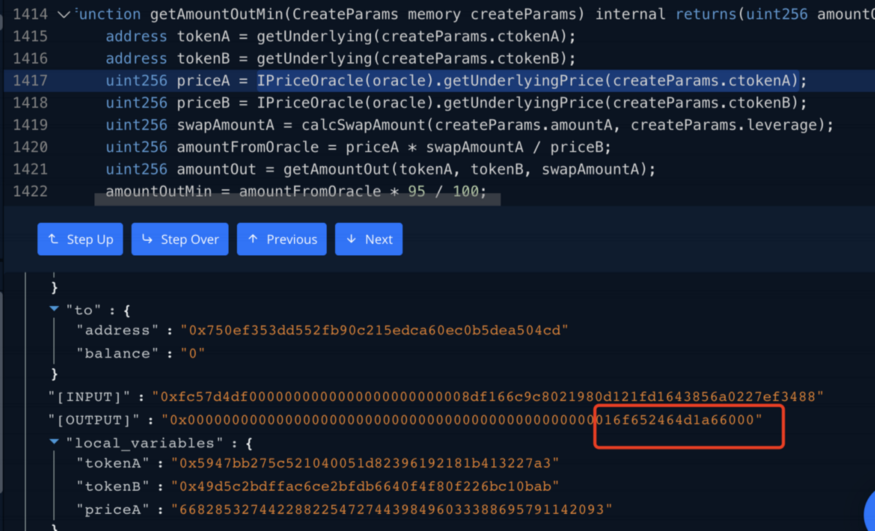
*bool isRightPrice = amountOut >= amountOutMin;*

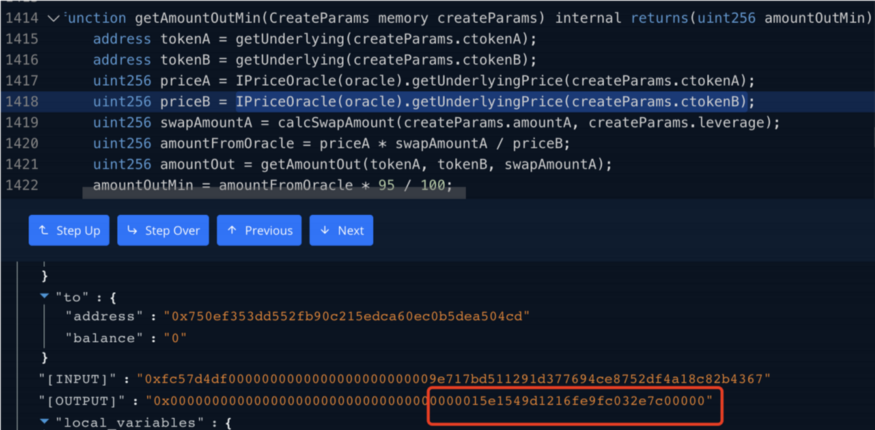
*确保swapAforB之后，得到的amountB价值大于等于原先amountA价值*

*require(isRightPrice,”price error”);*

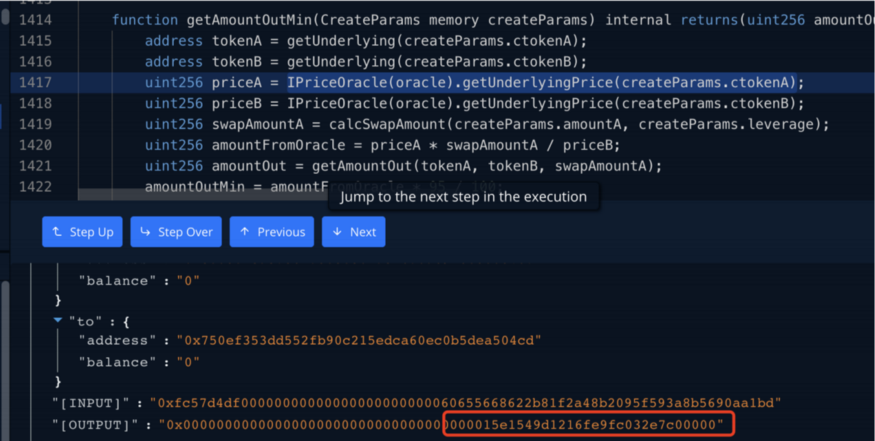
*}*

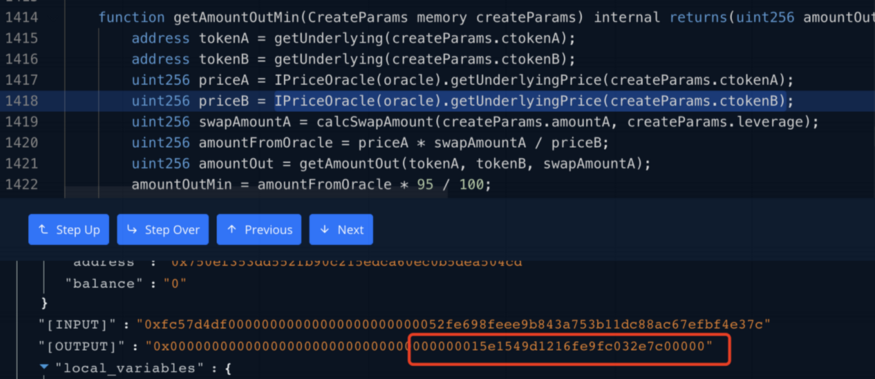
5.However, through on-chain records, when the oracle price is obtained, the obtained price decimals has not been processed. Therefore, if the decimals of TokenB is much greater than the decimals of TokenA, then there will be deviations in the calculation of the expected amount of exchangeable TokenB, **amountFromOracle = priceA \* swapAmountA / priceB** will be smaller than expected.





6. At the same time, in most attacks, the prices of TokenA and TokenB obtained by the oracle machine are equal, which shows that the price obtained by the oracle machine is wrong.





7. After communicating with the project party, the project party reported that the source of the price feed for the oracle machine only uses the price of the Pangolin pool, and the price of the pool fluctuates more than 3%, the oracle machine will refresh the price.

8. Therefore, the attacker manipulates the number of Pangolin’s tokens to make Vee Finance’s oracle machine to refresh the price. This directly caused the contract to obtain the wrong price from the oracle during the slippage check, which caused the slippage check to be bypassed.

**Fix Solution**

1. After the oracle machine obtains the token price, it should be processed with uniform decimals.

2. No support for tokens with a single source of price feed.

3. Modify the contract call check to: ***require(msg.sender == tx.origin)***

4. Whitelist restrictions on the target pair of margin trading.

5. Whitelist restrictions on oracle price feeding permissions.

**09/22 UPDATE**

The attacker forged cTokenB for leveraged transactions. Taking WBTC.e & XAVA as an example, *getUnderlying(createParams.ctokenB)* when performing getAmountOutMin is the XAVA address obtained by passing in the forged cTokenB from the attacker. But when the price is obtained through *getUnderlyingPrice(createParams.ctokenB)*, if the cToken is not in the list supported by the oracle, then the underlying price of cTokenB will be obtained through the oracle’s *getTokenConfigByUnderlying(CErc20(cToken).underlying())*. The underlying price at this time is taken by WBTC.e.

In summary, the attacker used the cToken forgery issue and the decimals processing issue of Oracle price to attack.