SharkTeam

Permission verification vulnerability: Analysis of the SushiSwap Attack

Apr 11,2023

SharkTeam, a leading blockchain security service team, offers smart contract audit services for developers. To satisfy the demands of different clients, thesmart contract audit services provide both manual auditing and automated auditing.

We implement almost 200 auditing contents that cover four aspects: high-level language layer, virtual machine layer, blockchain layer, and business logiclayer, ensuring that smart contracts are completely guaranteed and Safe.

On April 9, 2023, Beijing time, SushiSwap's RouteProcessor2 contract had a permission verification vulnerability, and the attacker has made a profit of about 3.3 million US dollars.

SharkTeam made a technical analysis of this incident and summarized the security precautions, hoping that subsequent projects can take this as a warning and build a security defense line for the blockchain industry.

1. Incident analysis

This attack included multiple transactions, and one transaction was taken as an example for analysis.

Attacker address: *0x719cdb61e217de6754ee8fc958f2866d61d565cf*

Attack contract: *0x000000c0524f353223d94fb76efab586a2ff8664*

Vulnerable contract: 0x044b75f554b886A065b9567891e45c79542d7357

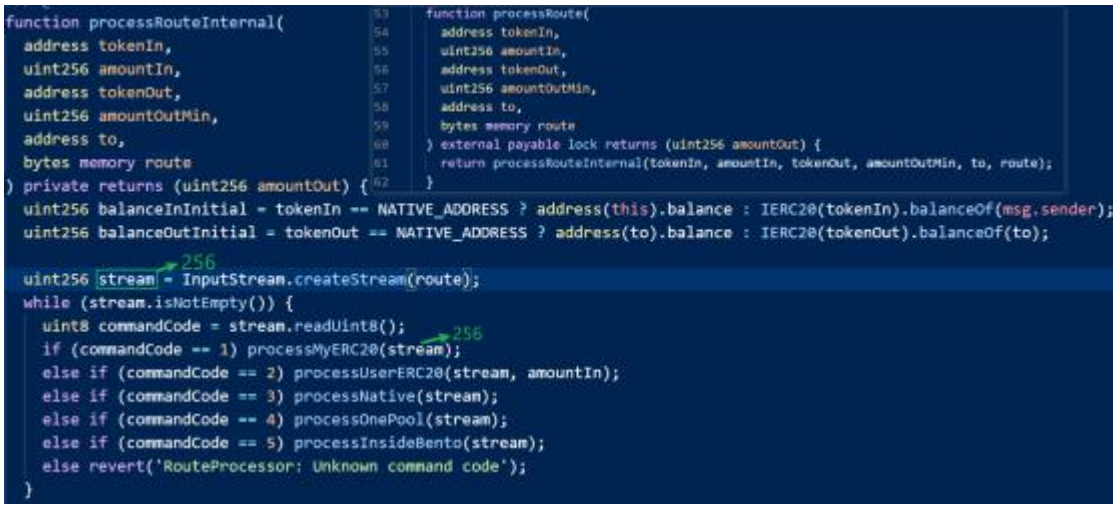
Attack transactions:

0xea3480f1f1d1f0b32283f8f282ce16403fe22ede35c0b71a732193e56c5c45e8

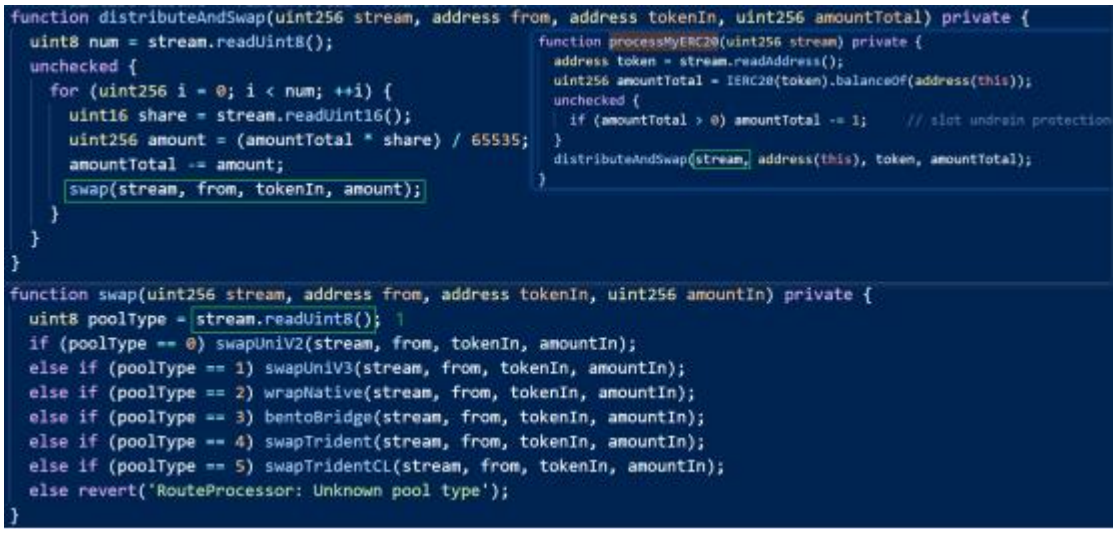
The execution flow of the attack transaction:

(1) First, the attacker (0x719cdb61) calls the processRoute function of the vulnerable contract (0x044b75f5).

(2) In the processRoute function, call processMyERC20 through the input parameter route.



(3) The stream parameter controlled by the attacker is called through a series of function calls such as processMyERC20 -> distributeAndSwap -> swap, and finally calls the swapUniV3 function.



(4) In the swapUniV3 function, the pool value is parsed from the stream parameter value controlled by the attacker, which is actually the attack contract. The parsed pool value is assigned to lastCalledPool. At this time, lastCalledPool is the malicious value controlled by the attacker (that is, the attack contract), and then the swap function defined by the pool, which is the attack contract, is called.

```
function swapUniV3(uint256 stream, address from, address tokenIn, uint256 amountIn) private {
  address pool = stream.readAddress();
  bool zeroForOne = stream.readUint8() > 0;
  address recipient = stream.readAddress();

  lastCalledPool = pool;
  IUniswapV3Pool(pool).swap(
    recipient,
    zeroForOne,
    int256(amountIn),
    zeroForOne ? MIN_SQRT_RATIO + 1 : MAX_SQRT_RATIO - 1,
    abi.encode(tokenIn, from)
  );
  require(lastCalledPool == IMPOSSIBLE_POOL_ADDRESS, 'RouteProcessor.swapUniV3: unexpected'); // Just to be sure
}
```

(5) The uniswapV3SwapCallback function in the vulnerable contract was called back through the swap function, and 100 ETHs were successfully extracted.

```
0x044b75f554b886A065b9567891e45c79542d7357.uniswapV3SwapCallback                [X]
Trace Path: 0.6.5.7
Type: call
Gas Used: 20233
Input Data [⌅] [▤]:
```

0xfa461e330000000000000000000000000000000000000000000000056bc75e2d631000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000600000000000000000000000000000000000000000000000000000000000000040000000000000000000000000c02aaa39b223fe8d0a0e5c4f27ead9083c756cc200000000000000000000000031d3243cfb54b34fc9c73e1cb1137124bd6b13e1

```
Output Data [⌅] [▤]:
0x
Function:
function uniswapV3SwapCallback(int256 arg0, int256 arg1, bytes arg2)
Decoded Inputs:
    arg0: 100000000000000000000
    arg1: 0
    arg2: 0x000000000000000000000000c02aaa39b223fe8d0a0e5c4f27ead9083c756cc200000000000000000000000031d3243cfb54b34fc9c73e1cb1137
Decoded Outputs:
```

Vulnerability analysis:

In the swapUniV3 function, the pool value comes from the stream parameter value maliciously constructed by the attacker, which is actually the attack contract, and then calls the custom swap function in the attack contract.


```
function swapUniV3(uint256 stream, address from, address tokenIn, uint256 amountIn) private {
  address pool = stream.readAddress();
  bool zeroForOne = stream.readUint8() > 0;
  address recipient = stream.readAddress();

  lastCalledPool = pool;
  IUniswapV3Pool(pool).swap(
    recipient,
    zeroForOne,
    int256(amountIn),
    zeroForOne ? MIN_SQRT_RATIO + 1 : MAX_SQRT_RATIO - 1,
    abi.encode(tokenIn, from)
  );
  require(lastCalledPool == IMPOSSIBLE_POOL_ADDRESS, 'RouteProcessor.swapUniV3: unexpected'); // Just to be sure
}
```

In the swap function, the uniswapV3SwapCallback function in the vulnerable contract is called back.

```
function uniswapV3SwapCallback(
  int256 amount0Delta,
  int256 amount1Delta,
  bytes calldata data
) external {
  require(msg.sender == lastCalledPool, 'RouteProcessor.uniswapV3SwapCallback: call from unknown source');
  lastCalledPool = IMPOSSIBLE_POOL_ADDRESS;
  (address tokenIn, address from) = abi.decode(data, (address, address));
  int256 amount = amount0Delta > 0 ? amount0Delta : amount1Delta;
  require(amount > 0, 'RouteProcessor.uniswapV3SwapCallback: not positive amount');

  if (from == address(this)) IERC20(tokenIn).safeTransfer(msg.sender, uint256(amount));
   else IERC20(tokenIn).safeTransferFrom(from, msg.sender, uint256(amount));
}
```

Since lastCalledPool is set to the pool value in the swapUniV3 function (that is, the address of the attack contract), when the uniswapV3SwapCallback function is called back, msg.sender == lastCalledPool is true, directly bypassing the authority check on whether msg.sender is a real pool address , and passed the parameters through the parameter construction token, stealing the WETH authorized by the account to the vulnerable contract that has not been used. Then set lastCalledPool to the constant IMPOSSIBLE_POOL_ADDRESS, thus passing the last require check of lastCalledPool==IMPOSSIBLE_POOL_ADDRESS in the swapUniV3 function.

Vulnerability Summary:

The root cause of this incident is that the vulnerable contract (RouteProcessor2) did not check whether the callback caller was a V3 pool deployed by the uniswapV3 factory, but only checked whether it was the original lastCalledPool value, resulting in the theft of too much unused funds authorized to the RouteProcessor2 contract for the account .

2. Security Recommendations

In response to this attack, we should follow the following precautions during the development process:

(1) In each business module where the user interacts with the contract, the principle of least authority should be followed, and the actual amount of a single transaction should be reasonably authorized to avoid the loss of account funds caused by excessive amount authorization.

(2) In a callback function similar to uniswapV3SwapCallback, it should be verified whether the caller is the V3 pool deployed by uniswapV3 factory.

About Us

Our vision is to improve security globally. We believe that by building this security barrier, we can significantly improve lives around the world.SharkTeam composes of members with many years of cyber security experiences and blockchain, team members are based in Suzhou, Beijing, Nanjing and Silicon Valley, proficient in the underlying theories of blockchain and smart contracts, and we provide comprehensive services including threat modeling, smart contract auditing, emergency response, etc. SharkTeam has established strategic and long-term cooperations with key players in many areas of the blockchain ecosystem, such as Huobi Global, OKX, polygon, Polkadot, imToken, ChainIDE, etc

Twitter：https://twitter.com/sharkteamorg

Discord：https://discord.gg/jGH9xXCjDZ

Telegram：https://t.me/sharkteamorg

web：https://www.sharkteam.org/

SharkTeam

In Math, We Trust!

https://sharkteam.org

https://t.me/sharkteamorg

https://twitter.com/sharkteamorg