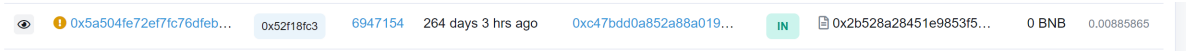
Uranium Finance的算术漏洞

攻击合约地址:0x2b528a28451e9853F51616f3B0f6D82Af8bEA6Ae

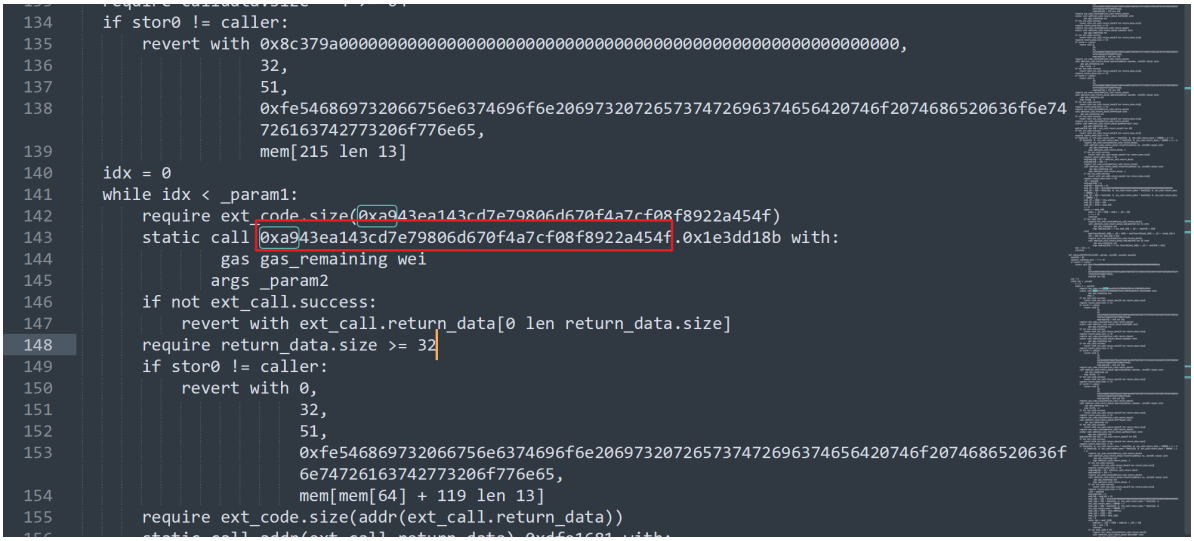
分析

Uranium项目中的合约漏洞出现再UraniumPair.sol合约中的swap函数中，这个漏洞会导致任何人可以随意的转出合约中的数字资产，而只需要付出一点点的代价。

通过查看最早的攻击交易（https://www.bscscan.com/txs?a=0x2b528a28451e9853F51616f3B0f6D82Af8bEA6Ae）：



得到攻击者调用的合约方法为0x52f18fc3。这是合约方法编码后的值，从反编译代码中寻找这个编码后的合约方法，可以找到这个合约攻击项目方的合约地址，也就是Uranium项目所在的地址：0xa943ea143cd7e79806d670f4a7cf08f8922a454f



```
134     if stor0 != caller:
135         revert with 0x8c379a000000000000000000000000000000000000000000000000000000000000000,
136                     32,
137                     51,
138                     0xfe546869732066756e6374696f6e206973207265737472696374656420746f2074686520636f6e74
                        726163742773206f776e65,
139                     mem[215 len 13]
140     idx = 0
141     while idx < _param1:
142         require ext_code.size(0xa943ea143cd7e79806d670f4a7cf08f8922a454f)
143         static call 0xa943ea143cd7e79806d670f4a7cf08f8922a454f.0x1e3dd18b with:
144                gas gas_remaining wei
145              args _param2
146         if not ext_call.success:
147             revert with ext_call.return_data[0 len return_data.size]
148         require return_data.size >= 32
149         if stor0 != caller:
150             revert with 0,
151                         32,
152                         51,
153                         0xfe546869732066756e6374696f6e206973207265737472696374656420746f2074686520636f6f
                            6e747261637427732066f776e65,
154                         mem[mem[64] + 119 len 13]
155         require ext_code.size(addr(ext_call.return_data))
```

出问题的是UraniumPair.sol合约中swap代码（https://www.bscscan.com/address/0xa943ea143cd7e79806d670f4a7cf08f8922a454f#code）：

```
    // this low-level function should be called from a contract which performs important safety checks
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
        require(amount0Out > 0 || amount1Out > 0, 'UraniumSwap: INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'UraniumSwap: INSUFFICIENT_LIQUIDITY');

        uint balance0;
        uint balance1;
        { // scope for _token{0,1}, avoids stack too deep errors
        address _token0 = token0;
        address _token1 = token1;
        require(to != _token0 && to != _token1, 'UraniumSwap: INVALID_TO');
        if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer tokens
        if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer tokens
        if (data.length > 0) IUraniumCallee(to).pancakeCall(msg.sender, amount0Out, amount1Out, data);
        balance0 = IERC20(_token0).balanceOf(address(this));
        balance1 = IERC20(_token1).balanceOf(address(this));
        }
        uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
        require(amount0In > 0 || amount1In > 0, 'UraniumSwap: INSUFFICIENT_INPUT_AMOUNT');
        { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
        uint balance0Adjusted = balance0.mul(10000).sub(amount0In.mul(16));
        uint balance1Adjusted = balance1.mul(10000).sub(amount1In.mul(16));
        require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000**2), 'UraniumSwap: K');
        }

        _update(balance0, balance1, _reserve0, _reserve1);
        emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
    }

    // force balances to match reserves
    function skim(address to) external lock {
        address _token0 = token0; // gas savings
        address _token1 = token1; // gas savings
        _safeTransfer(_token0, to, IERC20(_token0).balanceOf(address(this)).sub(reserve0));
        _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(this)).sub(reserve1));
    }
```

在UniswapV2Pair.sol中

```
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
        require(amount0Out > 0 || amount1Out > 0, 'UniswapV2: INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'UniswapV2: INSUFFICIENT_LIQUIDITY');

        uint balance0;
        uint balance1;
        { // scope for _token{0,1}, avoids stack too deep errors
        address _token0 = token0;
        address _token1 = token1;
        require(to != _token0 && to != _token1, 'UniswapV2: INVALID_TO');
        if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer tokens
        if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer tokens
        if (data.length > 0) IUniswapV2Callee(to).uniswapV2Call(msg.sender, amount0Out, amount1Out, data);
        balance0 = IERC20(_token0).balanceOf(address(this));
        balance1 = IERC20(_token1).balanceOf(address(this));
        }
        uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) : 0;
        require(amount0In > 0 || amount1In > 0, 'UniswapV2: INSUFFICIENT_INPUT_AMOUNT');
        { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
        uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
        uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
        require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000**2), 'UniswapV2: K');
        }

        _update(balance0, balance1, _reserve0, _reserve1);
        emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
    }

```

可以看到swap中，最后是一个10的8次方数和一个10的6次方数的比较，这是一个几乎是恒等的判断，这意味着，只要按照一定的套路不断的执行swap函数，就可以清空这个合约中所有的数字资产。

在UniswapV2Pair.sol合约中，写法是相同的，但是是两个10 的6次方进行比较。

这导致能够将1wei的输入代币换成输出代币总余额的98%。

因此造成此次事件的原因应该是项目方更新升级这个合约时，忘记了将后面的1000的2次方改为10000的2次方。

参考链接

https://www.cet.com.cn/xwsd/2842689.shtml

https://www.sharkteam.org/report/analysis/20210722001A.pdf