KNOWNSEC
Blockchain Lab

Smart Contract Audit Report

Version description

The revision	Date	Revised	Version
Write documentation	**20211126**	**KNOWNSEC Blockchain Lab**	**V1.0**

Document information

Title	Version	Document Number	Type
Hotpot Funds V3 Smart Contract Audit Report	**V1.0**	0e0851f0a849434a89193db957b2c170	**Open to project team**

Statement

KNOWNSEC Blockchain Lab only issues this report for facts that have occurred or existed before the issuance of this report, and assumes corresponding responsibilities for this. KNOWNSEC Blockchain Lab is unable to determine the security status of its smart contracts and is not responsible for the facts that will occur or exist in the future. The security audit analysis and other content made in this report are only based on the documents and information provided to us by the information provider as of the time this report is issued. KNOWNSEC Blockchain Lab 's assumption: There is no missing, tampered, deleted or concealed information. If the information provided is missing, tampered with, deleted, concealed or reflected in the actual situation, KNOWNSEC Blockchain Lab shall not be liable for any losses and adverse effects caused thereby.

Directory

1. Summarize

The effective test time of this report is **from November 16, 2021 to November 26, 2021**. During this period, the security and standardization of **the fund pool and fund pool controller code of the Hotpot Funds V3 smart contract** will be audited and reviewed. Use this as the statistical basis for the report.

The scope of this smart contract security audit does not include external contract calls, new attack methods that may appear in the future, and code after contract upgrades or tampering. (With the development of the project, the smart contract may add a new pool , New functional modules, new external contract calls, etc.), does not include front-end security and server security.

In this audit report, engineers conducted a comprehensive analysis of the common vulnerabilities of smart contracts (Chapter 6). **The smart contract code of the Hotpot Funds V3** is comprehensively assessed as **PASS**.

Since the testing is under non-production environment, all codes are the latest version. In addition, the testing process is communicated with the relevant engineer, and testing operations are carried out under the controllable operational risk to avoid production during the testing process, such as: Operational risk, code security risk.

KNOWNSEC Attest information:

classification	information
report number	0e0851f0a849434a89193db957b2c170
report query link	https://attest.im/attestation/searchResult?qurey=0e0851f0a849434a89193db957b2c170

2. Item information

2.1. Item description

Decentralized exchanges provide services for transactions by pooling liquidity. In theory, anyone can become a liquidity provider, but in reality, providing liquidity efficiently requires professional knowledge, in-depth data analysis, and corresponding automated tools. The original intention of the Hotpot Fund is to create valuable liquidity income by merging users' funds, managed by a professional fund team; to create valuable liquidity income under the premise of open source code, transparent operation, and user fund security.

2.2. The project's website

https://www.hotpot.fund/

2.3. White Paper

https://www.hotpot.fund/docs/White_Paper_en_V2.pdf

2.4. Review version code

https://github.com/HotPotFund/HotPotFundsV3

HotPotV3FundController.sol:https://etherscan.io/address/0xb440bd39870a94ba1131c6182ca5fba589d5449e#code

HotPotV3FundFactory.sol:https://etherscan.io/address/0xe9cf1fd8d9d804ef3ce6754776144b86c93efb8d#code

HotPot.sol:https://etherscan.io/address/0x615D8e5e1344B36A95F6ecd8e6CDA020E84dc25b#code

2.5. Contract file and Hash/contract deployment address

The contract documents	MD5
HotPotV3Fund.sol	5466b5ebba70e0b79542c7b5bc0190d7
Position.sol	95011c0d8554c9934d1d4bba31d73f95
FixedPoint64.sol	ee141a06c9382a6798b8c6c4c4fa90f7
PathPrice.sol	c41ee38f77254cddd75a97bb4b23a435
Array2D.sol	9f5ac7801a4a952b08425076800c3fc0
HotPotV3FundController.sol	43ba5aa371438cadbc27d6afcfe82e8f
HotPotV3FundFactory.sol	d49d1d6dedd4ece3e5f34c443b76f17b
HotPotV3FundDeployer.sol	80805e7d395dd7e0318d929d9fe478c0
Multicall.sol	50304727e75e3e5b9ceda24ce2bf2002
HotPotV3FundERC20.sol	e1049ffef73564dfbbef75d6885812e4
IHotPotV3FundFactory.sol	23a1bdda1cc008dfa5147a6639e27425
IMulticall.sol	c1b84afbc4676429f8e6d31b2d2900f2
IMulticall.sol	c1b84afbc4676429f8e6d31b2d2900f2
IHotPotV3FundERC20.sol	319725b2d04c579b829af746e886991d
IHotPotV3Fund.sol	7320f66f9fc016ef7275b77a70aa5685

IManagerActions.sol	8599cd9643971e4ea465aa676d5e9f85
IControllerEvents.sol	623e6f9b1ee724a50d47ff1d7d57e071
IControllerState.sol	035cc19f9602d2698d95d977af4835c3
IGovernanceActions.sol	8508a595617fe4dab9de9ec94dbd6fd1
IHotPotV3FundDeployer.sol	73c307b86d8b46911626c796369d9c7c
IHotPotV3FundController.sol	70cb2fe2345286b88c0548641bc1dda8
IHotPot.sol	74ca33b4030b551c329286d354d2e3bb
IWETH9.sol	4732f0afb7238d649338cce6e41cb4e5
IHotPotV3FundEvents.sol	1dffe1d43a6cb361fca1c0a36a299199
IHotPotV3FundManagerActions.sol	4d3b0cb7eacfd93da070b16533bffc08
IHotPotV3FundUserActions.sol	77d6561630d8fde9cd74abeb5e690185
IHotPotV3FundState.sol	25d26dd0b12298818e042c7192981fdb

3. External visibility analysis

3.1. HotPotV3Fund contracts

HotPotV3Fund					
funcName	**visibility**	**state changes**	**decorator**	**payable reception**	**instructions**
deposit	external	Ture	---	---	---
_deposit	internal	Ture	---	---	---
_assetsOfPool	internal	False	---	---	---
withdraw	external	Ture	checkDeadline(deadline), nonReentrant	---	---
poolsLength	external	False	---	---	---
positionsLength	external	False	---	---	---
setPath	external	Ture	onlyController	---	---
uniswapV3MintCallback	external	Ture	---	---	---
init	external	Ture	onlyController	---	---
add	external	Ture	onlyController	---	---
sub	external	Ture	onlyController	---	---
move	external	Ture	onlyController	---	---

assetsOfPosition	public	False	---	---	---
assetsOfPool	public	False	---	---	---
totalAssets	public	False	---	---	---
_assetsOfPool_assetsOfPool	internal	False	---	---	---

3.2. HotPotV3FundController contracts

HotPotV3FundController					
funcName	**visibility**	**state changes**	**decorator**	**payable reception**	**instructions**
maxPriceImpact	external	False	---	---	---
maxSqrtSlippage	external	False	---	---	---
setHarvestPath	external	True	onlyGovernance	---	---
setMaxPriceImpact	external	True	onlyGovernance	---	---
setMaxSqrtSlippage	external	True	onlyGovernance	---	---
harvest	external	True	---	---	---
setGovernance	external	True	onlyGovernance	---	---
setVerifiedToken	external	True	onlyGovernance	---	---
setPath	external	True	onlyManager(fund)	---	---

init	external	True	checkDeadline(deadline), onlyManager(fund)	---	---
add	external	True	checkDeadline(deadline), onlyManager(fund)	---	---
sub	external	True	checkDeadline(deadline), onlyManager(fund)	---	---
mov	external	True	checkDeadline(deadline), onlyManager(fund)	---	---

4. Code vulnerability analysis

4.1. Summary description of the audit results

Audit results			
audit project	**audit content**	**condition**	**description**
Business security detection	Extract function	Pass	After testing, there is no security issue.
	Harvest function	Pass	After testing, there is no security issue.
	Slippage check	Pass	After testing, there is no security issue.
	Increase or decrease in liquidity	Pass	After testing, there is no security issue.
	Fund inquiry	Pass	After testing, there is no security issue.
Code basic vulnerability detection	Compiler version security	Pass	After testing, there is no security issue.
	Redundant code	Pass	After testing, there is no security issue.
	Use of safe arithmetic library	Pass	After testing, there is no security issue.
	Not recommended encoding	Pass	After testing, there is no security issue.
	Reasonable use of require/assert	Pass	After testing, there is no security issue.
	fallback function safety	Pass	After testing, there is no security issue.
	tx.origin authentication	Pass	After testing, there is no security issue.
	Owner permission control	Pass	After testing, there is no security issue.
	Gas consumption detection	Pass	After testing, there is no security issue.
	call injection attack	Pass	After testing, there is no security issue.

	Low-level function safety	Pass	After testing, there is no security issue.
	Vulnerability of additional token issuance	Pass	After testing, there is no security issue.
	Access control defect detection	Pass	After testing, there is no security issue.
	Numerical overflow detection	Pass	After testing, there is no security issue.
	Arithmetic accuracy error	Pass	After testing, there is no security issue.
	Wrong use of random number detection	Pass	After testing, there is no security issue.
	Unsafe interface use	Pass	After testing, there is no security issue.
	Variable coverage	Pass	After testing, there is no security issue.
	Uninitialized storage pointer	Pass	After testing, there is no security issue.
	Return value call verification	Pass	After testing, there is no security issue.
	Transaction order dependency detection	Pass	After testing, there is no security issue.
	Timestamp dependent attack	Pass	After testing, there is no security issue.
	Denial of service attack detection	Pass	After testing, there is no security issue.
	Fake recharge vulnerability detection	Pass	After testing, there is no security issue.
	Reentry attack detection	Pass	After testing, there is no security issue.

	Replay attack detection	Pass	After testing, there is no security issue.
	Rearrangement attack detection	Pass	After testing, there is no security issue.

5. Business security detection

5.1. Extract function【Pass】

Audit analysis: Perform a security audit on the extraction function (withdraw) logic in the HotPotV3Fund.sol contract. The extraction purpose is (withdrawing a specified share of the local currency), and the amountMin and deadline parameters, as well as the price slippage and price impact limits, are added. The parameters are checked for legitimacy, and whether there are design flaws in the logic design for the withdrawal of the designated share of the local currency, and whether there is a reentry attack, etc. The method use permission is: external, which is a normal business requirement.

```
function withdraw(uint share, uint amountMin, uint deadline) external override checkDeadline(deadline) nonReentrant returns(uint amount) {
        uint balance = balanceOf[msg.sender];
        require(share > 0 && share <= balance, "ISA");
        uint investment = FullMath.mulDiv(investmentOf[msg.sender], share, balance);

        address fToken = token;
        // Construct the amounts array
        uint value = IERC20(fToken).balanceOf(address(this));
        uint _totalAssets = value;
        uint[][] memory amounts = new uint[][](pools.length);
        for(uint i=0; i<pools.length; i++){
            uint _amount;
            (_amount, amounts[i]) = _assetsOfPool(i);
            _totalAssets = _totalAssets.add(_amount);
        }
```

```
amount = FullMath.mulDiv(_totalAssets, share, totalSupply);
// Withdraw funds from the position from large to small.
if(amount > value) {
    uint remainingAmount = amount.sub(value);
    while(true) {
        // Take the largest position index number
        (uint poolIndex, uint positionIndex, uint desirableAmount) = amounts.max();
        if(desirableAmount == 0) break;

        if(remainingAmount <= desirableAmount){
            positions[poolIndex][positionIndex].subLiquidity(Position.SubParams({
                proportionX128: FullMath.mulDiv(remainingAmount, DIVISOR,
desirableAmount),
                pool: pools[poolIndex],
                token: fToken,
                uniV3Router: uniV3Router,
                uniV3Factory: uniV3Factory,
                maxSqrtSlippage: 10001,
                maxPriceImpact: 10001
            }), sellPath);
            break;
        }
        else {
            positions[poolIndex][positionIndex].subLiquidity(Position.SubParams({
                    proportionX128: DIVISOR,
                    pool: pools[poolIndex],
                    token: fToken,
                    uniV3Router: uniV3Router,
                    uniV3Factory: uniV3Factory,
                    maxSqrtSlippage: 10001,
                    maxPriceImpact: 10001
                }), sellPath);
            remainingAmount = remainingAmount.sub(desirableAmount);
```

```
                    amounts[poolIndex][positionIndex] = 0;
                }
            }
            /// When @dev withdraws funds from the liquidity pool, the liquidity is withdrawn proportionally, and all tokensOwed have been withdrawn, so the fund's local currency balance at this time will exceed the amount that users can withdraw.
            value = IERC20(fToken).balanceOf(address(this));
            // If the calculated value is greater than the actual fetched value
            if(amount > value)
                amount = value;
            // If it is the last person to withdraw
            else if(totalSupply == share)
                amount = value;
        }
```

Security advice: None.

5.2. Harvest function【Pass】

Audit analysis: Perform a security audit on the Harvest function logic in the HotPotV3FundController.sol contract. Its extraction purpose is (harvesting designated tokens). Compared with the previous version, it is modified to verify the price slippage through path calculation and check whether the parameters are legal Check whether the relevant logic design is reasonable, etc. The method use permission is: external, which is a normal business requirement.

```
function harvest(address token, uint amount) external override returns(uint burned) {
        bytes memory path = harvestPath[token]; //knownsec// Get token path
        PathPrice.verifySlippage(path, uniV3Factory, maxPIS & 0xffff); //knownsec// Verify slippage
```

```
        uint value = amount <= IERC20(token).balanceOf(address(this)) ? amount : IERC20(token).balanceOf(address(this));
        TransferHelper.safeApprove(token, uniV3Router, value);

        ISwapRouter.ExactInputParams memory args = ISwapRouter.ExactInputParams({
            path: path,
            recipient: address(this),
            deadline: block.timestamp,
            amountIn: value,
            amountOutMinimum: 0
        });
        burned = ISwapRouter(uniV3Router).exactInput(args);
        IHotPot(hotpot).burn(burned);
        emit Harvest(token, amount, burned);
    }
```

Security advice: None.

5.3. Slippage check【Pass】

Audit analysis: The slippage verification function is implemented in the verifySlippage function in the PathPrice.sol library contract file. It is used to calculate whether the slippage difference between the exchange current price and the oracle price does not exceed a given exchange path and maximum slippage. Maximum slippage.

```
library PathPrice {
    using Path for bytes;

  /// @notice Get the square root of the current price of the target token
      /// @param path conversion path
      /// @return sqrtPriceX96 The square root of the price (X 2^96), the price of tokenOut / tokenIn for a given exchange path
```

```
function getSqrtPriceX96(
        bytes memory path,
        address uniV3Factory
) internal view returns (uint sqrtPriceX96){
        require(path.length > 0, "IPL");

        sqrtPriceX96 = FixedPoint96.Q96;
        uint _nextSqrtPriceX96;
        uint32[] memory secondAges = new uint32[](2);
        secondAges[0] = 0;
        secondAges[1] = 1;
        while (true) {
                (address tokenIn, address tokenOut, uint24 fee) = path.decodeFirstPool();
                IUniswapV3Pool pool =
IUniswapV3Pool(PoolAddress.computeAddress(uniV3Factory, PoolAddress.getPoolKey(tokenIn,
tokenOut, fee)));

                (_nextSqrtPriceX96,,,,,,) = pool.slot0();
                sqrtPriceX96 = tokenIn > tokenOut
                        ? FullMath.mulDiv(sqrtPriceX96, FixedPoint96.Q96, _nextSqrtPriceX96)
                        : FullMath.mulDiv(sqrtPriceX96, _nextSqrtPriceX96, FixedPoint96.Q96);

                // decide whether to continue or terminate
                if (path.hasMultiplePools())
                        path = path.skipToken();
                else
                        break;
        }
}

/// @notice Get the square root of the price of the target token oracle
  /// @param path conversion path
  /// @return sqrtPriceX96Last The square root of the oracle price (X 2^96), the price of
```

```
tokenOut / tokenIn for a given exchange path
    function getSqrtPriceX96Last(
        bytes memory path,
        address uniV3Factory
    ) internal view returns (uint sqrtPriceX96Last){
        require(path.length > 0, "IPL");

        sqrtPriceX96Last = FixedPoint96.Q96;
        uint _nextSqrtPriceX96;
        uint32[] memory secondAges = new uint32[](2);
        secondAges[0] = 0;
        secondAges[1] = 1;
        while (true) {
            (address tokenIn, address tokenOut, uint24 fee) = path.decodeFirstPool();
            IUniswapV3Pool pool = IUniswapV3Pool(PoolAddress.computeAddress(uniV3Factory, PoolAddress.getPoolKey(tokenIn, tokenOut, fee)));

            // sqrtPriceX96Last
            (int56[] memory tickCumulatives,) = pool.observe(secondAges);
            _nextSqrtPriceX96 = TickMath.getSqrtRatioAtTick(int24(tickCumulatives[0] - tickCumulatives[1]));
            sqrtPriceX96Last = tokenIn > tokenOut
                ? FullMath.mulDiv(sqrtPriceX96Last, FixedPoint96.Q96, _nextSqrtPriceX96)
                : FullMath.mulDiv(sqrtPriceX96Last, _nextSqrtPriceX96, FixedPoint96.Q96);

            // decide whether to continue or terminate
            if (path.hasMultiplePools())
                path = path.skipToken();
            else
                break;
        }
    }
```

```
    /// @notice verify whether the transaction slippage meets the conditions
      /// @param path conversion path
      /// @param uniV3Factory uniswap v3 factory
      /// @param maxSqrtSlippage maximum slippage, maximum value: 1e4
      /// @return current price
    function verifySlippage(
        bytes memory path,
        address uniV3Factory,
        uint32 maxSqrtSlippage
    ) internal view returns(uint) { //knownsec// Check slippage
        uint last = getSqrtPriceX96Last(path, uniV3Factory);
        uint current = getSqrtPriceX96(path, uniV3Factory);
        if(last > current) require(current > FullMath.mulDiv(maxSqrtSlippage, last, 1e4), "VS");
        return current;
    }
}
```

Security advice: None.

5.4. Increase or decrease in liquidity【Pass】

Audit analysis: The liquidity increase and decrease function is implemented by the addLiquidity function and subLiquidity function in the Position.sol library contract, which is used to add liquidity for investment and reduce the position LP with divestment.

```
function addLiquidity(
    Info storage self,
    AddParams memory params,
    mapping(address => bytes) storage sellPath,
    mapping(address => bytes) storage buyPath
) public returns(uint128 liquidity) {
    (int24 tickLower, int24 tickUpper) = (self.tickLower, self.tickUpper);
```

```
(uint160 sqrtPriceX96,,,,,,) = IUniswapV3Pool(params.pool).slot0();

SwapParams memory swapParams = SwapParams({
    amount: params.amount,
    amount0: params.amount0Max,
    amount1: params.amount1Max,
    sqrtPriceX96: sqrtPriceX96,
    sqrtRatioAX96: TickMath.getSqrtRatioAtTick(tickLower),
    sqrtRatioBX96: TickMath.getSqrtRatioAtTick(tickUpper),
    token: params.token,
    token0: IUniswapV3Pool(params.pool).token0(),
    token1: IUniswapV3Pool(params.pool).token1(),
    fee: IUniswapV3Pool(params.pool).fee(),
    uniV3Router: params.uniV3Router,
    uniV3Factory: params.uniV3Factory,
    maxSqrtSlippage: params.maxSqrtSlippage,
    maxPriceImpact: params.maxPriceImpact
});
(params.amount0Max, params.amount1Max) = computeSwapAmounts(swapParams, buyPath);

//Because of slippage, reload sqrtPriceX96
(sqrtPriceX96,,,,,,) = IUniswapV3Pool(params.pool).slot0();

// Estimate the actual liquidity
liquidity = LiquidityAmounts.getLiquidityForAmounts(sqrtPriceX96, swapParams.sqrtRatioAX96, swapParams.sqrtRatioBX96, params.amount0Max, params.amount1Max);

require(liquidity > 0, "LIZ");
(uint amount0, uint amount1) = IUniswapV3Pool(params.pool).mint(
    address(this),// LP recipient
```

```
            tickLower,
            tickUpper,
            liquidity,
            abi.encode(params.poolIndex)
        );

        // Process the token balance not added to the LP and exchange it back to the fund's local currency
        if(amount0 < params.amount0Max){
            if(swapParams.token0 != params.token){
                ISwapRouter(params.uniV3Router).exactInput(ISwapRouter.ExactInputParams({
                    path: sellPath[swapParams.token0],
                    recipient: address(this),
                    deadline: block.timestamp,
                    amountIn: params.amount0Max - amount0,
                    amountOutMinimum: 0
                }));
            }
        }
        if(amount1 < params.amount1Max){
            if(swapParams.token1 != params.token){
                ISwapRouter(params.uniV3Router).exactInput(ISwapRouter.ExactInputParams({
                    path: sellPath[swapParams.token1],
                    recipient: address(this),
                    deadline: block.timestamp,
                    amountIn: params.amount1Max - amount1,
                    amountOutMinimum: 0
                }));
            }
        }

        if(self.isEmpty) self.isEmpty = false;
}
```

```
......
function subLiquidity (
        Info storage self,
        SubParams memory params,
        mapping(address => bytes) storage sellPath
) public returns(uint amount) {
        address token0 = IUniswapV3Pool(params.pool).token0();
        address token1 = IUniswapV3Pool(params.pool).token1();
        uint sqrtPriceX96;
        uint sqrtPriceX96Last;
        uint amountOutMin;

        // Verify the slippage of the pool
        if(params.maxSqrtSlippage <= 1e4){
                // Slippage from t0 to t1
                (sqrtPriceX96,,,,,,) = IUniswapV3Pool(params.pool).slot0();
                uint32[] memory secondAges = new uint32[](2);
                secondAges[0] = 0;
                secondAges[1] = 1;
                (int56[] memory tickCumulatives,) = IUniswapV3Pool(params.pool).observe(secondAges);
                sqrtPriceX96Last = TickMath.getSqrtRatioAtTick(int24(tickCumulatives[0] - tickCumulatives[1]));
                if(sqrtPriceX96Last > sqrtPriceX96)
                        require(sqrtPriceX96 > params.maxSqrtSlippage * sqrtPriceX96Last / 1e4, "VS");// No overflow

                // Slippage from t1 to t0
                sqrtPriceX96 = FixedPoint96.Q96 * FixedPoint96.Q96 / sqrtPriceX96; // No overflow
                sqrtPriceX96Last = FixedPoint96.Q96 * FixedPoint96.Q96 / sqrtPriceX96Last;
                if(sqrtPriceX96Last > sqrtPriceX96)
                        require(sqrtPriceX96 > params.maxSqrtSlippage * sqrtPriceX96Last / 1e4, "VS"); // No overflow
```

```
    }

    // burn & collect
    (uint amount0, uint amount1) = burnAndCollect(self, params.pool, params.proportionX128);

    // t0 is converted into fund local currency
    if(token0 != params.token){
        if(amount0 > 0){
            bytes memory path = sellPath[token0];
            if(params.maxSqrtSlippage <= 1e4) {
                sqrtPriceX96 = PathPrice.verifySlippage(path, params.uniV3Factory, params.maxSqrtSlippage);
                amountOutMin = getAmountOutMin(sqrtPriceX96, params.maxPriceImpact, amount0);
            }
            amount = ISwapRouter(params.uniV3Router).exactInput(ISwapRouter.ExactInputParams({
                path: path,
                recipient: address(this),
                deadline: block.timestamp,
                amountIn: amount0,
                amountOutMinimum: amountOutMin
            }));
        }
    }

    // t1 is converted into the fund's local currency
    if(token1 != params.token){
        if(amount1 > 0){
            bytes memory path = sellPath[token1];
            if(params.maxSqrtSlippage <= 1e4) {
                sqrtPriceX96 = PathPrice.verifySlippage(path, params.uniV3Factory, params.maxSqrtSlippage);
```

```
                amountOutMin = getAmountOutMin(sqrtPriceX96, params.maxPriceImpact, amount1);
            }
            amount = amount.add(ISwapRouter(params.uniV3Router).exactInput(ISwapRouter.ExactInputParams({
                path: path,
                recipient: address(this),
                deadline: block.timestamp,
                amountIn: amount1,
                amountOutMinimum: amountOutMin
            })));
        }
    }
}
```

Security advice: None.

5.5. Fund inquiry【Pass】

Audit analysis: The fund query function is implemented by the assetsOfPool function and the assets function in the Position.sol library contract, which is used to query the assets of the liquidity pool and a certain position.

```
function assetsOfPool(
    Info[] storage self,
    address pool,
    address token,
    mapping(address => bytes) storage sellPath,
    address uniV3Factory
) public view returns (uint amount, uint[] memory) {
    uint[] memory amounts = new uint[](self.length);
    // Local variables are used to reduce sload consumption.
    AssetsParams memory params;
```

```
// Get the local currency prices of two tokens.
params.token0 = IUniswapV3Pool(pool).token0();
params.token1 = IUniswapV3Pool(pool).token1();
if(params.token0 != token){
    bytes memory path = sellPath[params.token0];
    if(path.length == 0) return(amount, amounts);
    params.sqrt0 = PathPrice.getSqrtPriceX96Last(path, uniV3Factory);
}
if(params.token1 != token){
    bytes memory path = sellPath[params.token1];
    if(path.length == 0) return(amount, amounts);
    params.sqrt1 = PathPrice.getSqrtPriceX96Last(path, uniV3Factory);
}

(params.sqrtPriceX96, params.tick, , , , , ) = IUniswapV3Pool(pool).slot0();
params.feeGrowthGlobal0X128 = IUniswapV3Pool(pool).feeGrowthGlobal0X128();
params.feeGrowthGlobal1X128 = IUniswapV3Pool(pool).feeGrowthGlobal1X128();

for(uint i=0; i < self.length; i++){
    Position.Info memory position = self[i];
    if(position.isEmpty) continue;
    bytes32 positionKey = keccak256(abi.encodePacked(address(this), position.tickLower,
position.tickUpper));
    // Get the number of assets of token0, token1
    (uint256 _amount0, uint256 _amount1) =
        getAssetsOfSinglePosition(
            AssetsOfSinglePosition({
                pool: pool,
                positionKey: positionKey,
                tickLower: position.tickLower,
                tickUpper: position.tickUpper,
                tickCurrent: params.tick,
                sqrtPriceX96: params.sqrtPriceX96,
```

```
                        feeGrowthGlobal0X128: params.feeGrowthGlobal0X128,
                        feeGrowthGlobal1X128: params.feeGrowthGlobal1X128
                    })
                );

            // Calculate cost currency assets.
            uint _amount;
            if(params.token0 != token){
                _amount = FullMath.mulDiv(
                    _amount0,
                    FullMath.mulDiv(params.sqrt0, params.sqrt0, FixedPoint64.Q64),
                    FixedPoint128.Q128);
            }
            else
                _amount = _amount0;

            if(params.token1 != token){
                _amount = _amount.add(FullMath.mulDiv(
                    _amount1,
                    FullMath.mulDiv(params.sqrt1, params.sqrt1, FixedPoint64.Q64),
                    FixedPoint128.Q128));
            }
            else
                _amount = _amount.add(_amount1);

            amounts[i] = _amount;
            amount = amount.add(_amount);
        }
        return(amount, amounts);
}

/// @notice Get a position, all assets measured in the fund's currency
/// @param pool transaction pool index number
```

```
/// @param token position index number
/// @return amount Asset quantity
function assets(
    Info storage self,
    address pool,
    address token,
    mapping(address => bytes) storage sellPath,
    address uniV3Factory
) public view returns (uint amount) {
    if(self.isEmpty) return 0;

    // No need to verify the existence of pool
    (uint160 sqrtPriceX96, int24 tick, , , , , ) = IUniswapV3Pool(pool).slot0();

    bytes32 positionKey = keccak256(abi.encodePacked(address(this), self.tickLower, self.tickUpper));

    // Get the number of assets of token0, token1
    (uint256 amount0, uint256 amount1) =
        getAssetsOfSinglePosition(
            AssetsOfSinglePosition({
                pool: pool,
                positionKey: positionKey,
                tickLower: self.tickLower,
                tickUpper: self.tickUpper,
                tickCurrent: tick,
                sqrtPriceX96: sqrtPriceX96,
                feeGrowthGlobal0X128: IUniswapV3Pool(pool).feeGrowthGlobal0X128(),
                feeGrowthGlobal1X128: IUniswapV3Pool(pool).feeGrowthGlobal1X128()
            })
        );

    // Calculate assets measured in local currency.
```

```
    if(amount0 > 0){
        address token0 = IUniswapV3Pool(pool).token0();
        if(token0 != token){
            uint sqrt0 = PathPrice.getSqrtPriceX96Last(sellPath[token0], uniV3Factory);
            amount = FullMath.mulDiv(
                amount0,
                FullMath.mulDiv(sqrt0, sqrt0, FixedPoint64.Q64),
                FixedPoint128.Q128);
        } else
            amount = amount0;
    }
    if(amount1 > 0){
        address token1 = IUniswapV3Pool(pool).token1();
        if(token1 != token){
            uint sqrt1 = PathPrice.getSqrtPriceX96Last(sellPath[token1], uniV3Factory);
            amount = amount.add(FullMath.mulDiv(
                amount1,
                FullMath.mulDiv(sqrt1, sqrt1, FixedPoint64.Q64),
                FixedPoint128.Q128));
        } else
            amount = amount.add(amount1);
    }
}
```

Security advice: None.

6. Code basic vulnerability detection

6.1. Compiler version security【Pass】

Check to see if a secure compiler version is used in the contract code implementation.

Detection results: After detection, the smart contract code has developed a compiler version of 0.7.6, there is no security issue.

Security advice: None.

6.2. Redundant code【Pass】

Check that the contract code implementation contains redundant code.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.3. Use of safe arithmetic library【Pass】

Check to see if the SafeMath security abacus library is used in the contract code implementation.

Detection results: The SafeMath security abacus library has been detected in the smart contract code and there is no such security issue.

Security advice: None.

6.4. Not recommended encoding【Pass】

Check the contract code implementation for officially uns recommended or deprecated coding methods.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.5. Reasonable use of require/assert【Pass】

Check the reasonableness of the use of require and assert statements in contract code implementations.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.6. Fallback function safety【Pass】

Check that the fallback function is used correctly in the contract code implementation.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.7. tx.origin authentication【Pass】

tx.origin is a global variable of Solidity that traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts makes contracts vulnerable to phishing-like attacks.z

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.8. Owner permission control【Pass】

Check that theowner in the contract code implementation has excessive permissions. For example, modify other account balances at will, and so on.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.9. Gas consumption detection【Pass】

Check that the consumption of gas exceeds the maximum block limit.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.10. call injection attack 【Pass】

When a call function is called, strict permission control should be exercised, or the function called by call calls should be written directly to call calls.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.11. Low-level function safety 【Pass】

Check the contract code implementation for security vulnerabilities in the use of call/delegatecall

The execution context of the call function is in the contract being called, while the execution context of the delegatecall function is in the contract in which the function is currently called.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.12. Vulnerability of additional token issuance 【Pass】

Check to see if there are functions in the token contract that might increase the total token volume after the token total is initialized.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.13. Access control defect detection 【Pass】

Different functions in the contract should set reasonable permissions, check whether the functions in the contract correctly use pubic, private and other keywords for visibility modification, check whether the contract is properly defined and use modifier access restrictions on key functions, to avoid problems caused by overstepping the authority.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.14. Numerical overflow detection 【Pass】

The arithmetic problem in smart contracts is the integer overflow and integer overflow, with Solidity able to handle up to 256 digits (2^256-1), and a maximum number increase of 1 will overflow to get 0. Similarly, when the number is an unsigned type, 0 minus 1 overflows to get the maximum numeric value.

Integer overflows and underflows are not a new type of vulnerability, but they are particularly dangerous in smart contracts. Overflow conditions can lead to incorrect results, especially if the likelihood is not anticipated, which can affect the reliability and safety of the program.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.15. Arithmetic accuracy error【Pass】

Solidity has a data structure design similar to that of a normal programming language, such as variables, constants, arrays, functions, structures, and so on, and there is a big difference between Solidity and a normal programming language - Solidity does not have floating-point patterns, and all of Solidity's numerical operations result in integers, without the occurrence of decimals, and without allowing the definition of decimal type data. Numerical operations in contracts are essential, and numerical operations are designed to cause relative errors, such as sibling operations: 5/2 x 10 x 20, and 5 x 10/2 x 25, resulting in errors, which can be greater and more obvious when the data is larger.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.16. Incorrect use of random numbers【Pass】

Random numbers may be required in smart contracts, and while the functions and variables provided by Solidity can access significantly unpredictable values, such as block.number and block.timestamp, they are usually either more public than they seem, or are influenced by miners, i.e. these random numbers are somewhat predictable, so malicious users can often copy it and rely on its unpredictability to attack the feature.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.17. Unsafe interface usage【Pass】

Check the contract code implementation for unsafe external interfaces, which can be controlled, which can cause the execution environment to be switched and control contract execution arbitrary code.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.18. Variable coverage【Pass】

Check the contract code implementation for security issues caused by variable overrides.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.19. Uninitialized storage pointer【Pass】

A special data structure is allowed in solidity as a strut structure, while local variables within the function are stored by default using stage or memory.

The existence of store (memory) and memory (memory) is two different concepts, solidity allows pointers to point to an uninitialized reference, while uninitialized local stage causes variables to point to other stored variables, resulting in variable overrides, and even more serious consequences, and should avoid initializing the task variable in the function during development.

Detection results: After detection, the smart contract code does not have the problem.

Security advice: None.

6.20. Return value call verification【Pass】

This issue occurs mostly in smart contracts related to currency transfers, so it is also known as silent failed sending or unchecked sending.

In Solidity, there are transfer methods such as transfer(), send(), call.value(), which can be used to send tokens to an address, the difference being: transfer send failure will be throw, and state rollback; Call.value returns false when it fails to send, and passing all available gas calls (which can be restricted by incoming gas_value parameters) does not effectively prevent reentration attacks.

If the return values of the send and call.value transfer functions above are not checked in the code, the contract continues to execute the subsequent code, possibly with unexpected results due to token delivery failures.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.21. Transaction order dependency【Pass】

Because miners always get gas fees through code that represents an externally owned address (EOA), users can specify higher fees to trade faster. Since blockchain is public, everyone can see the contents of other people's pending transactions. This means that if a user submits a valuable solution, a malicious user can steal the solution and copy its transactions at a higher cost to preempt the original solution.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.22. Timestamp dependency attack【Pass】

Block timestamps typically use miners' local time, which can fluctuate over a range of about 900 seconds, and when other nodes accept a new chunk, they only need to verify that the timestamp is later than the previous chunk and has a local time error of less than 900 seconds. A miner can profit from setting the timestamp of a block to meet as much of his condition as possible.

Check the contract code implementation for key timestamp-dependent features.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.23. Denial of service attack【Pass】

Smart contracts that are subject to this type of attack may never return to normal operation. There can be many reasons for smart contract denial of service, including malicious behavior as a transaction receiver, the exhaustion of gas caused by the artificial addition of the gas required for computing functionality, the misuse of access control to access the private component of smart contracts, the exploitation of confusion and negligence, and so on.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.24. Fake recharge vulnerability【Pass】

The transfer function of the token contract checks the balance of the transfer initiator (msg.sender) in the if way, when the balances < value enters the else logic part and return false, and ultimately does not throw an exception, we think that only if/else is a gentle way of judging in a sensitive function scenario such as transfer is a less rigorous way of coding.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.25. Reentry attack detection【Pass】

The call.value() function in Solidity consumes all the gas it receives when it is used to send tokens, and there is a risk of re-entry attacks when the call to the call tokens occurs before the balance of the sender's account is actually reduced.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.26. Replay attack detection【Pass】

If the requirements of delegate management are involved in the contract, attention should be paid to the non-reusability of validation to avoid replay attacks

In the asset management system, there are often cases of entrustment management, the principal will be the assets to the trustee management, the principal to pay a certain fee to the trustee. This business scenario is also common in smart contracts.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.27. Rearrangement attack detection【Pass】

A reflow attack is an attempt by a miner or other party to "compete" with a smart contract participant by inserting their information into a list or mapping, giving an attacker the opportunity to store their information in a contract.

Detection results: After detection, there are no related vulnerabilities in the smart contract code.

Security advice: None.

7. Appendix A: Security Assessment of Contract Fund Management

Contract fund management		
The type of asset in the contract	**The function is involved**	**Security risks**
User token assets	deposit、withdraw、add、sub、move	**SAFE**

Check the security of the management of **digital currency assets** transferred by users in the business logic of the contract. Observe whether there are security risks that may cause the loss of customer funds, such as **incorrect recording, incorrect transfer, and backdoor** withdrawal of the **digital currency assets** transferred into the contract.

KNOWNSEC
Blockchain Lab

Official Website

www.knownseclab.com

E-mail

blockchain@knownsec.com

WeChat Official Account