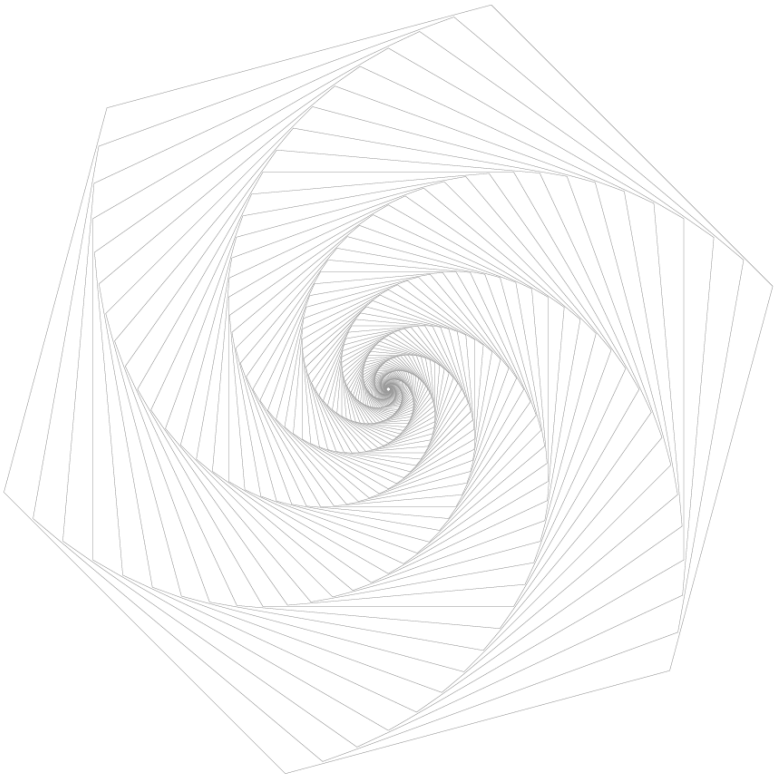
知道创宇

区块链安全实验室

智能合约审计报告



版本说明

修订内容	时间	修订者	版本号
编写文档	20211126	知道创宇区块链安全实验室	V1.0

文档信息

文档名称	文档版本	报告编号	保密级别
Hotpot Funds V3 智能合约审计报告	V1.0	0e0851f0a849434a89193db957b2c170	项目组公开

声明

创宇区块链安全实验室仅就本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后发生或存在的事实，创宇区块链安全实验室无法判断其智能合约安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向创宇区块链安全实验室提供的文件和资料。创宇区块链安全实验室假设:已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，创宇区块链安全实验室对由此而导致的损失和不利影响不承担任何责任。

目录

1. 综述

本次报告有效测试时间是从 2021 年 11 月 16 日开始到 2021 年 11 月 26 日结束，在此期间针对 **Hotpot Funds V3 智能合约的基金池、基金池控制器代码**安全性和规范性进行审计并以此作为报告统计依据。

本次智能合约安全审计的范围，不包含外部合约调用，不包含未来可能出现的新型攻击方式，不包含合约升级或篡改后的代码(随着项目方的发展，智能合约可能会增加新的 pool、新的功能模块，新的外部合约调用等)，不包含前端安全与服务器安全。

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第六章节）以及合约具体业务安全项进行了全面的分析，**综合评定为通过**。

由于本次测试过程在非生产环境下进行，所有代码均为最新备份，测试过程均与相关接口人进行沟通，并在操作风险可控的情况下进行相关测试操作，以规避测试过程中的生产运营风险、代码安全风险。

创宇存证信息:

类别	信息
报告编号	0e0851f0a849434a89193db957b2c170
报告查询链接	https://attest.im/attestation/searchResult?qurey=0e0851f0a849434a89193db957b2c170

2. 项目信息

2.1. 项目描述

去中心化交易所通过汇聚流动性为交易提供服务。理论上，任何人都可以成为流动性提供者，但实际上，高效提供流动性需要专业知识、深入的数据分析和相应的自动化工具。**Hotpot 基金**的初衷是通过合并用户的基金，创造有价值的流动性收益，由专业的基金团队管理；在开源代码、透明运营、用户资金安全的前提下创造有价值的流动性收益。

2.2. 项目官网

https://www.hotpot.fund/

2.3. 白皮书

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

2.4. 审核版本代码

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. 合约文件及哈希/合约部署地址

合约文件	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. 外部可见性分析

3.1. HotPotV3Fund 合约

HotPotV3Fund					
函数名	可见性	状态修改	修饰器	可支付接收	说明
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 合约

HotPotV3FundController					
函数名	**可见性**	**状态修改**	**修饰器**	**可支付接收**	**说明**
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. 代码漏洞分析

4.1. 审计结果汇总说明

审计结果			
审计项目	**审计内容**	**状态**	**描述**
业务安全性检测	**提取功能**	通过	经检测，不存在安全问题。
	Harvest 功能	通过	经检测，不存在安全问题。
	滑点校验	通过	经检测，不存在安全问题。
	流动性增减	通过	经检测，不存在安全问题。
	资金查询	通过	经检测，不存在安全问题。
代码基本漏洞检测	**编译器版本安全**	通过	经检测，不存在该安全问题。
	冗余代码	通过	经检测，不存在该安全问题。
	安全算数库的使用	通过	经检测，不存在该安全问题。
	不推荐的编码方式	通过	经检测，不存在该安全问题。
	require/assert 的合理使用	通过	经检测，不存在该安全问题。
	fallback 函数安全	通过	经检测，不存在该安全问题。
	tx.origin 身份验证	通过	经检测，不存在该安全问题。
	owner 权限控制	通过	经检测，不存在该安全问题。
	gas 消耗检测	通过	经检测，不存在该安全问题。
	call 注入攻击	通过	经检测，不存在该安全问题。
	低级函数安全	通过	经检测，不存在该安全问题。
	增发代币漏洞	通过	经检测，不存在该安全问题。
	访问控制缺陷检测	通过	经检测，不存在该安全问题。
	数值溢出检测	通过	经检测，不存在该安全问题。
	算数精度误差	通过	经检测，不存在该安全问题。
	错误使用随机数检测	通过	经检测，不存在该安全问题。

	不安全的外部调用	通过	经检测，不存在该安全问题。
	变量覆盖	通过	经检测，不存在该安全问题。
	未初始化的存储指针	通过	经检测，不存在该安全问题。
	返回值调用验证	通过	经检测，不存在该安全问题。
	交易顺序依赖检测	通过	经检测，不存在该安全问题。
	时间戳依赖攻击	通过	经检测，不存在该安全问题。
	拒绝服务攻击检测	通过	经检测，不存在该安全问题。
	假充值漏洞检测	通过	经检测，不存在该安全问题。
	重入攻击检测	通过	经检测，不存在该安全问题。
	重放攻击检测	通过	经检测，不存在该安全问题。
	重排攻击检测	通过	经检测，不存在该安全问题。

5. 业务安全性检测

5.1. 提取功能【通过】

审计分析：对 HotPotV3Fund.sol 合约中提取函数(**withdraw**)逻辑进行安全审计，其提取用途为(**提取指定份额本币**)，新增了 amountMin 和 deadline 参数以及价格滑点和价格影响限制，检查是否有对参数进行合法性校验，指定份额本币取出逻辑设计是否存在设计缺陷、是否存在重入攻击等。方法使用权限为权限为：external，**属于业务正常需求。**

```
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;
        // 构造 amounts 数组
        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);
        // 从大到小从头寸中撤资.
        if(amount > value) {
            uint remainingAmount = amount.sub(value);
```

```
            while(true) {
                // 取最大的头寸索引号
                (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;
                }
            }
            /// @dev 从流动池中撤资时，按比例撤流动性，同时 tokensOwed 已全部提取，
```

```
所以此时的基金本币余额会超过用户可提金额.
            value = IERC20(fToken).balanceOf(address(this));
            // 如果计算值比实际取出值大
            if(amount > value)
                amount = value;
            // 如果是最后一个人 withdraw
            else if(totalSupply == share)
                amount = value;
        }
```

安全建议：无。

5.2. Harvest 功能【通过】

审计分析：对 HotPotV3FundController.sol 合约中 Harvest 函数逻辑进行安全审计，其提取用途为(**收获指定代币**)，相比上一版修改为通过路径计算验证价格滑点，检查是否有对参数进行合法性校验，相关逻辑设计是否合理等。方法使用权限为权限为：external，**属于业务正常需求。**

```
function harvest(address token, uint amount) external override returns(uint burned) {
        bytes memory path = harvestPath[token]; //knownsec// 获取代币路径
        PathPrice.verifySlippage(path, uniV3Factory, maxPIS & 0xffff); //knownsec// 验证滑点
        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);
    }
```

安全建议：无。

5.3. 滑点校验【通过】

审计分析：滑点校验功能在 PathPrice.sol 库合约文件中的 verifySlippage 函数实现，用于在给定的兑换路径和最大滑点中，计算兑换现价与预言机价格的滑点差值是否不超过最大滑点。

```
library PathPrice {
    using Path for bytes;

    /// @notice 获取目标代币当前价格的平方根
    /// @param path 兑换路径
    /// @return sqrtPriceX96 价格的平方根(X 2^96)，给定兑换路径的 tokenOut / tokenIn 的价格
    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 获取目标代币预言机价格的平方根
    /// @param path 兑换路径
    /// @return sqrtPriceX96Last 预言机价格的平方根(X 2^96)，给定兑换路径的 tokenOut / tokenIn 的价格
    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 验证交易滑点是否满足条件
    /// @param path 兑换路径
    /// @param uniV3Factory uniswap v3 factory
    /// @param maxSqrtSlippage 最大滑点,最大值: 1e4
    /// @return 当前价
    function verifySlippage(
        bytes memory path,
        address uniV3Factory,
        uint32 maxSqrtSlippage
    ) internal view returns(uint) { //knownsec// 校验滑点
```

```
        uint last = getSqrtPriceX96Last(path, uniV3Factory);
        uint current = getSqrtPriceX96(path, uniV3Factory);
        if(last > current) require(current > FullMath.mulDiv(maxSqrtSlippage, last, 1e4), "VS");
        return current;
    }
}
```

安全建议：无。

5.4. 流动性增减【通过】

审计分析：流动性增减功能由 Position.sol 库合约中的 addLiquidity 函数和 subLiquidity 函数实现，用于投资添加流动性和撤资减少头寸 LP。

```
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);

        //因为滑点，重新加载 sqrtPriceX96
        (sqrtPriceX96,,,,,) = IUniswapV3Pool(params.pool).slot0();

        //推算实际的 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)
        );

        //处理没有添加进 LP 的 token 余额，兑换回基金本币
        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;

        // 验证本池子的滑点
```

```
    if(params.maxSqrtSlippage <= 1e4){
        // t0 到 t1 的滑点
        (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");// 不会出现溢出

        // t1 到 t0 的滑点
        sqrtPriceX96 = FixedPoint96.Q96 * FixedPoint96.Q96 / sqrtPriceX96; // 不会出现溢出
        sqrtPriceX96Last = FixedPoint96.Q96 * FixedPoint96.Q96 / sqrtPriceX96Last;
        if(sqrtPriceX96Last > sqrtPriceX96)
            require(sqrtPriceX96 > params.maxSqrtSlippage * sqrtPriceX96Last / 1e4, "VS"); // 不会出现溢出
    }

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

    // t0 兑换成基金本币
    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 兑换成基金本币
    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
            })));
        }
```

```
    }
}
```

安全建议：无。

5.5. 资金查询【通过】

审计分析：资金查询功能由 Position.sol 库合约中的 assetsOfPool 函数和 assets 函数实现，用于查询流动池的资产和某个头寸。

```
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);
    // 局部变量都是为了减少 ssload 消耗.
    AssetsParams memory params;
    // 获取两种 token 的本币价格.
    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));
    // 获取 token0, token1 的资产数量
    (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
            })
        );

    // 计算成本币资产.
    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 获取某个头寸，以基金本币衡量的所有资产
    /// @param pool 交易池索引号
    /// @param token 头寸索引号
    /// @return amount 资产数量
    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;

        // 不需要校验 pool 是否存在
        (uint160 sqrtPriceX96, int24 tick, , , , ) = IUniswapV3Pool(pool).slot0();
```

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

// 获取 token0, token1 的资产数量
(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()
        })
    );

// 计算以本币衡量的资产.
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);
        }
    }
```

安全建议：无。

6. 代码基本漏洞检测

6.1. 编译器版本安全【通过】

检查合约代码实现中是否使用了安全的编译器版本。

检测结果：经检测，智能合约代码中制定了编译器版本 0.7.6，不存在该安全问题。

安全建议：无。

6.2. 冗余代码【通过】

检查合约代码实现中是否包含冗余代码。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.3. 安全算数库的使用【通过】

检查合约代码实现中是否使用了 SafeMath 安全算数库。

检测结果：经检测，智能合约代码中已使用 SafeMath 安全算数库，不存在该安全问题。

安全建议：无。

6.4. 不推荐的编码方式【通过】

检查合约代码实现中是否有官方不推荐或弃用的编码方式。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.5. require/assert 的合理使用【通过】

检查合约代码实现中 require 和 assert 语句使用的合理性。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.6. fallback 函数安全【通过】

检查合约代码实现中是否正确使用 fallback 函数。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.7. tx.origin 身份验证【通过】

tx.origin 是 Solidity 的一个全局变量，它遍历整个调用栈并返回最初发送调用（或事务）的帐户的地址。在智能合约中使用此变量进行身份验证会使合约容易受到类似网络钓鱼的攻击。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.8. owner 权限控制【通过】

检查合约代码实现中的 owner 是否具有过高的权限。例如，任意修改其他账户余额等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.9. gas 消耗检测【通过】

检查 gas 的消耗是否超过区块最大限制。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.10. call 注入攻击【通过】

call 函数调用时，应该做严格的权限控制，或直接写死 call 调用的函数。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.11. 低级函数安全【通过】

检查合约代码实现中低级函数（call/delegatecall）的使用是否存在安全漏洞

call 函数的执行上下文是在被调用的合约中；而 delegatecall 函数的执行上下文是在当前调用该函数的合约中。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.12. 增发代币漏洞【通过】

检查在初始化代币总量后，代币合约中是否存在可能使代币总量增加的函数。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.13. 访问控制缺陷检测【通过】

合约中不同函数应设置合理的权限，检查合约中各函数是否正确使用了 public、private 等关键词进行可见性修饰，检查合约是否正确定义并使用了 modifier 对关键函数进行访问限制，避免越权导致的问题。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.14. 数值溢出检测【通过】

智能合约中的算数问题是指整数溢出和整数下溢，Solidity 最多能处理 256 位的数字（2^256-1），最大数字增加 1 会溢出得到 0。同样，当数字为无符号类型时，0 减去 1 会下溢得到最大数字值。

整数溢出和下溢不是一种新类型的漏洞，但它们在智能合约中尤其危险。溢出情况会导致不正确的结果，特别是如果可能性未被预期，可能会影响程序的可靠性和安全性。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.15. 算术精度误差【通过】

Solidity 作为一门编程语言具备和普通编程语言相似的数据结构设计，比如：变量、常量、数组、函数、结构体等等，Solidity 和普通编程语言也有一个较大的区别——Solidity 没有浮点型，且 Solidity 所有的数值运算结果都只会是整数，不会出现小数的情况，同时也不允许定义小数类型数据。合约中的数值运算必不

可少，而数值运算的设计有可能造成相对误差，例如同级运算：5/2*10=20,而5*10/2=25，从而产生误差，在数据更大时产生的误差也会更大，更明显。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.16. 错误使用随机数【通过】

智能合约中可能需要使用随机数，虽然 Solidity 提供的函数和变量可以访问明显难以预测的值，如 block.number 和 block.timestamp，但是它们通常或者比看起来更公开，或者受到矿工的影响，即这些随机数在一定程度上是可预测的，所以恶意用户通常可以复制它并依靠其不可预知性来攻击该功能。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.17. 不安全的外部调用【通过】

检查合约代码实现中是否使用了不安全的外部接口，接口可控可能导致执行环境被切换，控制合约执行任意代码。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.18. 变量覆盖【通过】

检查合约代码实现中是否存在变量覆盖导致的安全问题。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.19. 未初始化的储存指针【通过】

在 solidity 中允许一个特殊的数据结构为 struct 结构体，而函数内的局部变量默认使用 storage 或 memory 储存。

而存在 storage(存储器)和 memory(内存)是两个不同的概念，solidity 允许指针指向一个未初始化的引用，而未初始化的局部 stroage 会导致变量指向其他储存变量，导致变量覆盖，甚至其他更严重的后果，在开发中应该避免在函数中初始化 struct 变量。

检测结果：经检测，智能合约代码不存在该问题。

安全建议：无。

6.20. 返回值调用验证【通过】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

在 Solidity 中存在 transfer()、send()、call.value()等转币方法，都可以用于向某一地址发送代币，其区别在于： transfer 发送失败时会 throw，并且进行状态回滚；只会传递 2300gas 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300gas 供调用，防止重入攻击；call.value 发送失败时会返回 false；传递所有可用 gas 进行调用（可通过传入 gas_value 参数进行限制），不能有效防止重入攻击。

如果在代码中没有检查以上 send 和 call.value 转币函数的返回值，合约会继

续执行后面的代码，可能由于代币发送失败而导致意外的结果。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.21. 交易顺序依赖【通过】

由于矿工总是通过代表外部拥有地址（EOA）的代码获取 gas 费用，因此用户可以指定更高的费用以便更快地开展交易。由于区块链是公开的，每个人都可以看到其他人未决交易的内容。这意味着，如果某个用户提交了一个有价值的解决方案，恶意用户可以窃取该解决方案并以较高的费用复制其交易，以抢占原始解决方案。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.22. 时间戳依赖攻击【通过】

数据块的时间戳通常来说都是使用矿工的本地时间，而这个时间大约能有 900 秒的范围波动，当其他节点接受一个新区块时，只需要验证时间戳是否晚于之前的区块并且与本地时间误差在 900 秒以内。一个矿工可以通过设置区块的时间戳来尽可能满足有利于他的条件来从中获利。

检查合约代码实现中是否存在有依赖于时间戳的关键功能。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.23. 拒绝服务攻击【通过】

遭受该类型攻击的智能合约可能永远无法恢复正常工作状态。导致智能合约拒绝服务的原因可能有很多种，包括在作为交易接收方时的恶意行为，人为增加计算功能所需 gas 导致 gas 耗尽，滥用访问控制访问智能合约的 private 组件，利用混淆和疏忽等等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.24. 假充值漏洞【通过】

在代币合约的 transfer 函数对转账发起人(msg.sender)的余额检查用的是 if 判断方式，当 balances[msg.sender] < value 时进入 else 逻辑部分并 return false，最终没有抛出异常，我们认为仅 if/else 这种温和的判断方式在 transfer 这类敏感函数场景中是一种不严谨的编码方式。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.25. 重入攻击检测【通过】

Solidity 中的 call.value()函数在被用来发送代币的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送代币的操作发生在实际减少发送者账户的余额之前时，就会存在重入攻击的风险。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.26. 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击

在资产管理体系中，常有委托管理的情况，委托人将资产给受托人管理，委托人支付一定的费用给受托人。这个业务场景在智能合约中也比较普遍。。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.27. 重排攻击检测【通过】

重排攻击是指矿工或其他方试图通过将自己的信息插入列表(list)或映射(mapping)中来与智能合约参与者进行“竞争”，从而使攻击者有机会将自己的信息存储到合约中。

检测结果：经检测，智能合约代码中不存在相关漏洞。

安全建议：无。

7. 附录 A：合约资金管理安全评估

合约资金管理		
合约内资产类型	**涉及函数**	**安全风险**
用户代币资产	deposit、withdraw、add、sub、move	安全

检查合约业务逻辑中用户转入**数字货币资产**的**管理安全**性。观察是否存在转入合约的**数字货币资产**被**错误记录、错误转出、后门提现**等易引起客户资金损失的安全风险。

KNOWNSEC
Blockchain Lab

官方网站

www.knownseclab.com

邮箱

blockchain@knownsec.com

微信公众号