知道创宇

密级 ★ | 项目组公开

智能合约审计报告

安全状态

安全

★ ★ ★ ★ ★

主测人： 知道创宇区块链安全研究团队

版本说明

修订内容	时间	修订者	版本号
编写文档	20210723	知道创宇区块链安全研究团队	V1.0

文档信息

文档名称	文档版	报告编号	保密级别
HotpotFunds V2 智能合约审计报告	V1.0	8dfbdc445a49421ea471d0c9b961ebfb	项目组公开

声明

创宇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提供的文件和资料。创宇假设: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 6 月 11 日开始到 2021 年 7 月 23 日结束，在此期间针对 **HotpotFunds V2 智能合约代码**的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，不包括外部合约调用，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（见第四章节）进行了全面的分析，同时对业务逻辑层面进行审计，未发现存在相关安全风险，故对此合约综合评定为 **通过**。

本次智能合约安全审计结果： 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。

本次审计的报告信息：

报告编号：8dfbdc445a49421ea471d0c9b961ebfb

报告查询地址链接:

https://attest.im/attestation/searchResult?qurey=8dfbdc445a49421ea471d0c9b961ebfb

本次审计的目标信息：

条目	描述
项目名称	Hotpot Funds V2
合约地址	HotPotV2Controller 0xe366d53F07af38cE012aeA484747E40D513d000E HotPotV2Factory

	0xc2D232A140a5B308295d5F7F84f5Ec6f02d42fFC HotPot 0x615d8e5e1344b36a95f6ecd8e6cda020e84dc25b
代码类型	以太坊智能合约代码
代码语言	Solidity

合约主要文件哈希：

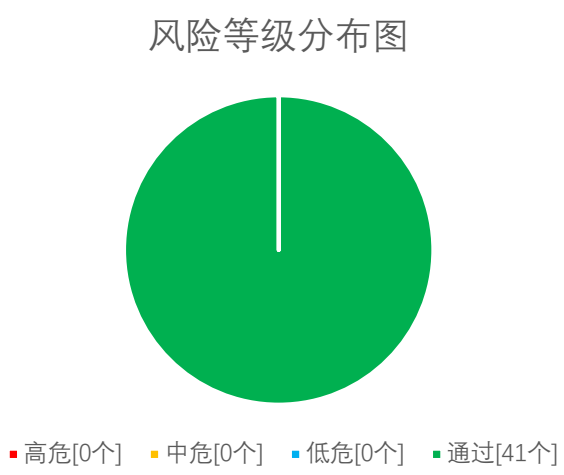
合约文件	MD5
HotPotV2FundERC20.sol	42A5D3E2D4C0EAE6697BA1AEE00DAC41
Multicall.sol	50304727E75E3E5B9CEDA24CE2BF2002
Array2D.sol	9F5AC7801A4A952B08425076800C3FC0
HotPotV2Fund.sol	E210103BAE4F596B29BCBEE850423519
HotPotV2FundController.sol	BD4FCB0FABF088BC326F13882E166CE8
HotPotV2FundDeployer.sol	5C98FB79D2EC3837A8E90A63C1B6A6A0
HotPotV2FundFactory.sol	53772B491AE059E5FF0267AF4F309BCF
Position.sol	CE7052E1BD3F303EA5C7F52B6CBC3A2F
PathPrice.sol	2E0A0E65E01D70B3725A9D01A94896CF

2. 代码漏洞分析

2.1 漏洞等级分布

本次漏洞风险按等级统计：

安全风险等级个数统计表			
高危	中危	低危	通过
0	0	0	41



2.2 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	设置受信任 Token	通过	经检测，不存在该安全问题。
	Harvest 逻辑设	通过	经检测，不存在该安全问题。
	更新 Harvest 路径	通过	经检测，不存在该安全问题。
	更新治理账户地址	通过	经检测，不存在该安全问题。
	设置代币交易路径	通过	经检测，不存在该安全问题。
	初始化头寸逻辑	通过	经检测，不存在该安全问题。
	投资指定头寸逻辑	通过	经检测，不存在该安全问题。
	撤资指定头寸逻辑	通过	经检测，不存在该安全问题。
	调整头寸投资逻辑	通过	经检测，不存在该安全问题。
	基金本币存入逻辑	通过	经检测，不存在该安全问题。
	提取指定份额本币	通过	经检测，不存在该安全问题。
	创建基金逻辑设计	通过	经检测，不存在该安全问题。
	Receive 逻辑设计	通过	经检测，不存在该安全问题。
	setmaxHarvestSlippage	通过	经检测，不存在该安全问题。
编码安全	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.orgin 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。
	不安全的接口使用	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	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	通过	经检测，不存在该安全问题。
	假充值漏洞	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。

3. 业务安全性检测

3.1. 设置受信任 Token【通过】

对合约中设置受信任 Token 的相关逻辑进行安全审计，检查是否有对参数进行合法性校验、权限设计是否合理、是否有相应的事件记录与触发机制设计等。

审计结果：经审计，设置受信任 Token 相关逻辑设计合理无误。

```
/// @inheritdoc IGovernanceActions
function setVerifiedToken(address token, bool isVerified) external override onlyGovernance {
    verifiedToken[token] = isVerified;
    emit ChangeVerifiedToken(token, isVerified);
}
modifier onlyGovernance{
    require(msg.sender == governance, "OGC");
    _;
}
```

安全建议：无。

3.2. Harvest 逻辑设计【通过】

对合约中 Harvest 相关逻辑进行安全审计，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。

审计结果：经审计，Harvest 相关逻辑设计合理无误。

```
/// @inheritdoc IHotPotV2FundController
function harvest(address token, uint amount) external override returns(uint burned) {
    uint value = amount <= IERC20(token).balanceOf(address(this)) ? amount : IERC20(token).balanceOf(address(this));
    TransferHelper.safeApprove(token, uniV3Router, value);
    uint curPirce = PathPrice.getSqrtPriceX96(harvestPath[token], uniV3Factory, true);
```

```
        uint lastPrice = PathPrice.getSqrtPriceX96(harvestPath[token], uniV3Factory, false);
        if(lastPrice > curPirce) {
            lastPrice = FullMath.mulDiv(lastPrice, lastPrice, FixedPoint96.Q96);
            require(FullMath.mulDiv(lastPrice - FullMath.mulDiv(curPirce, curPirce, FixedPoint96.Q96), 100, lastPrice) <= maxHarvestSlippage, "MHS");
        }

        ISwapRouter.ExactInputParams memory args = ISwapRouter.ExactInputParams({
            path: harvestPath[token],
            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。

3.3. 更新 Harvest 路径【通过】

对合约中更新 Harvest 路径相关逻辑设计进行安全审计，检查是否有对参数进行合法性校验、权限设计是否合理、是否有相应的事件记录与触发机制设计等。

审计结果：经审计，未发现存在相关安全风险。

```
    /// @inheritdoc IGovernanceActions
    function setHarvestPath(address token, bytes memory path) external override onlyGovernance {
        bytes memory _path = path;
        while (true) {
            (address tokenIn, address tokenOut, uint24 fee) = path.decodeFirstPool();
```

```
            // pool is exist
            address pool = IUniswapV3Factory(uniV3Factory).getPool(tokenIn, tokenOut, fee);
            require(pool != address(0), "PIE");
            // at least 2 observations
            (,,,uint16 observationCardinality,,,) = IUniswapV3Pool(pool).slot0();
            require(observationCardinality >= 2, "OC");

            if (path.hasMultiplePools()) {
                path = path.skipToken();
            } else {
                //最后一个交易对：输入 WETH9，输出 hotpot
                require(tokenIn == WETH9 && tokenOut == hotpot, "IOT");
                break;
            }
        }
        harvestPath[token] = _path;
        emit SetHarvestPath(token, _path);
    }
```

安全建议：无。

3.4. 更新治理账户地址【通过】

对合约中更新治理账户地址相关逻辑进行安全审计，检查是否有对参数进行合法性校验、权限设计是否合理、是否有相应的事件记录与触发机制设计等。

审计结果：经审计，未发现存在相关安全风险。

```
    /// @inheritdoc IGovernanceActions
    function setGovernance(address account) external override onlyGovernance {
        require(account != address(0));
        governance = account;
        emit SetGovernance(account);
    }
```

```
modifier onlyGovernance{
    require(msg.sender == governance, "OGC");
    _;
}
```

安全建议：无。

3.5. 设置代币交易路径【通过】

对合约中设置代币交易路径相关逻辑进行安全审计，检查是否有对参数进行合法性校验、权限设计是否合理、是否有相应的事件记录与触发机制设计等。

审计结果：经审计，未发现存在相关安全风险。

```
/// @inheritdoc IManagerActions
function setPath(
    address fund,
    address distToken,
    bytes memory path
) external override onlyManager(fund){
    require(verifiedToken[distToken]);

    address fundToken = IHotPotV2Fund(fund).token();
    bytes memory _path = path;
    bytes memory _reverse;
    (address tokenIn, address tokenOut, uint24 fee) = path.decodeFirstPool();
    _reverse = abi.encodePacked(tokenOut, fee, tokenIn);
    bool isBuy;
    // 第一个 tokenIn 是基金 token，那么就是 buy 路径
    if(tokenIn == fundToken){
        isBuy = true;
    }
    // 如果是 sellPath，第一个需要是目标代币
    else{
```

```
            require(tokenIn == distToken);
        }

        while (true) {
            require(verifiedToken[tokenIn], "VIT");
            require(verifiedToken[tokenOut], "VOT");
            // pool is exist
            address pool = IUniswapV3Factory(uniV3Factory).getPool(tokenIn, tokenOut, fee);
            require(pool != address(0), "PIE");
            // at least 2 observations
            (,,,uint16 observationCardinality,,,) = IUniswapV3Pool(pool).slot0();
            require(observationCardinality >= 2, "OC");

            if (path.hasMultiplePools()) {
                path = path.skipToken();
                (tokenIn, tokenOut, fee) = path.decodeFirstPool();
                _reverse = abi.encodePacked(tokenOut, fee, _reverse);
            } else {
                /// @dev  如果是 buy,  最后一个 token 要是目标代币;
                /// @dev  如果是 sell,  最后一个 token 要是基金 token.
                if(isBuy)
                    require(tokenOut == distToken, "OID");
                else
                    require(tokenOut == fundToken, "OIF");
                break;
            }
        }
        emit SetPath(fund, distToken, _path);
        if(!isBuy) (_path, _reverse) = (_reverse, _path);
        IHotPotV2Fund(fund).setPath(distToken, _path, _reverse);
    }
    modifier onlyManager(address fund){
        require(msg.sender == IHotPotV2Fund(fund).manager(), "OMC");
```

```
        _;
    }
    /// @inheritdoc IHotPotV2FundManagerActions
    function setPath(
        address distToken,
        bytes memory buy,
        bytes memory sell
    ) external override onlyController{
        // 要修改 sellPath，需要先清空相关 pool 头寸资产
        if(sellPath[distToken].length > 0){
            for(uint i = 0; i < pools.length; i++){
                IUniswapV3Pool pool = IUniswapV3Pool(pools[i]);
                if(pool.token0() == distToken || pool.token1() == distToken){
                    (uint amount,) = _assetsOfPool(i);
                    require(amount == 0, "AZ");
                }
            }
        }

        TransferHelper.safeApprove(distToken, uniV3Router, 0);
        TransferHelper.safeApprove(distToken, uniV3Router, 2**256-1);
        buyPath[distToken] = buy;
        sellPath[distToken] = sell;
    }
```

安全建议：无。

3.6. 初始化头寸逻辑【通过】

对合约中初始化头寸逻辑设计进行安全审计，检查是否有对参数进行合法性校验、权限设计是否合理、初始化头寸逻辑设计是否合理等。

审计结果：经审计，初始化头寸逻辑设计合理无误，未发现存在安全风险。

```
    /// @inheritdoc IManagerActions
    function init(
        address fund,
        address token0,
        address token1,
        uint24 fee,
        int24 tickLower,
        int24 tickUpper,
        uint amount
    ) external override onlyManager(fund){
        IHotPotV2Fund(fund).init(token0, token1, fee, tickLower, tickUpper, amount);
    }

    modifier onlyManager(address fund){
        require(msg.sender == IHotPotV2Fund(fund).manager(), "OMC");
        _;
    }
    /// @inheritdoc IHotPotV2FundManagerActions
    function init(
        address token0,
        address token1,
        uint24 fee,
        int24 tickLower,
        int24 tickUpper,
        uint amount
    ) external override onlyController{
        // 1、检查 pool 是否有效
        require(tickLower < tickUpper && token0 < token1, "ITV");
        address pool = IUniswapV3Factory(uniV3Factory).getPool(token0, token1, fee);
        require(pool != address(0), "ITF");
        int24 tickspacing = IUniswapV3Pool(pool).tickSpacing();
        require(tickLower % tickspacing == 0, "TLV");
        require(tickUpper % tickspacing == 0, "TUV");
```

```
        // 2、添加流动池
        bool hasPool = false;
        uint poolIndex;
        for(uint i = 0; i < pools.length; i++){
            // 存在相同的流动池
            if(pools[i] == pool) {
                hasPool = true;
                poolIndex = i;
                for(uint positionIndex = 0; positionIndex < positions[i].length; positionIndex++) {
                    // 存在相同的头寸，退出
                    if(positions[i][positionIndex].tickLower == tickLower &&
positions[i][positionIndex].tickUpper == tickUpper)
                        revert();
                }
                break;
            }
        }
        if(!hasPool) {
            pools.push(pool);
            positions.push();
            poolIndex = pools.length - 1;
        }

        //3、新增头寸
        positions[poolIndex].push(Position.Info({
            isEmpty: true,
            tickLower: tickLower,
            tickUpper: tickUpper
        }));

        //4、投资
        if(amount > 0){
```

```
            address fToken = token;
            require(IERC20(fToken).balanceOf(address(this)) >= amount, "ATL");
            Position.Info storage position = positions[poolIndex][positions[poolIndex].length - 1];
            position.addLiquidity(Position.AddParams({
                poolIndex: poolIndex,
                pool: pool,
                amount: amount,
                amount0Max: 0,
                amount1Max: 0,
                token: fToken,
                uniV3Router: uniV3Router,
                uniV3Factory: uniV3Factory
            }), sellPath, buyPath);
        }
    }
```

安全建议：无。

3.7. 投资指定头寸逻辑【通过】

对合约中投资指定头寸逻辑设计进行审计，检查是否有对参数进行合法性检查，投资指定头寸逻辑设计是否合理等。

审计结果：经审计，投资指定头寸逻辑设计合理无误，未发现相关安全风险。

```
    /// @inheritdoc IManagerActions
    function add(
        address fund,
        uint poolIndex,
        uint positionIndex,
        uint amount,
        bool collect
    ) external override onlyManager(fund){
        IHotPotV2Fund(fund).add(poolIndex, positionIndex, amount, collect);
```

```
}
/// @inheritdoc IHotPotV2FundManagerActions
function add(
        uint poolIndex,
        uint positionIndex,
        uint amount,
        bool collect
) external override onlyController {
        require(IERC20(token).balanceOf(address(this)) >= amount, "ATL");
        require(poolIndex < pools.length, "IPL");
        require(positionIndex < positions[poolIndex].length, "IPS");

        uint amount0Max;
        uint amount1Max;
        Position.Info storage position = positions[poolIndex][positionIndex];
        address pool = pools[poolIndex];
        // 需要复投?
        if(collect) (amount0Max, amount1Max) = position.burnAndCollect(pool, 0);

        position.addLiquidity(Position.AddParams({
                poolIndex: poolIndex,
                pool: pool,
                amount: amount,
                amount0Max: amount0Max,
                amount1Max: amount1Max,
                token: token,
                uniV3Router: uniV3Router,
                uniV3Factory: uniV3Factory
        }), sellPath, buyPath);
}
/// @notice brun 指定头寸的 LP，并取回 2 种代币
/// @param pool UniswapV3Pool
/// @param proportionX128 burn 所占份额
```

```
    /// @return amount0  获得的 token0 数量
    /// @return amount1  获得的 token1 数量
    function burnAndCollect(
        Info storage self,
        address pool,
        uint proportionX128
    ) public returns(uint amount0, uint amount1) {
        require(proportionX128 <= DIVISOR, "PTL");

        // 如果是空头寸，直接返回 0,0
        if(self.isEmpty == true) return(amount0, amount1);

        int24 tickLower = self.tickLower;
        int24 tickUpper = self.tickUpper;

        IUniswapV3Pool _pool = IUniswapV3Pool(pool);
        if(proportionX128 > 0) {
            (uint sumLP, , , , ) = _pool.positions(PositionKey.compute(address(this), tickLower, tickUpper));
            uint subLP = FullMath.mulDiv(proportionX128, sumLP, DIVISOR);

            _pool.burn(tickLower, tickUpper, uint128(subLP));
            (amount0, amount1) = _pool.collect(address(this), tickLower, tickUpper, type(uint128).max, type(uint128).max);

            if(sumLP == subLP) self.isEmpty = true;
        }
        //为 0 表示只提取手续费
        else {
            _pool.burn(tickLower, tickUpper, 0);
            (amount0, amount1) = _pool.collect(address(this), tickLower, tickUpper, type(uint128).max, type(uint128).max);
        }
```

```
    }
    /// @notice  添加 LP 到指定 Position
    /// @param self Position.Info
    /// @param params  投资信息
    /// @param sellPath sell token 路径
    /// @param buyPath buy token 路径
    function addLiquidity(
        Info storage self,
        AddParams memory params,
        mapping(address => bytes) storage sellPath,
        mapping(address => bytes) storage buyPath
    ) public {
        (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
        });
        (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
        uint128 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;
    }
    /// @notice 根据基金本币数量以及收集的手续费数量，计算投资指定头寸两种代币的分布.
    function computeSwapAmounts(
        SwapParams memory params,
        mapping(address => bytes) storage buyPath
    ) internal returns(uint amount0Max, uint amount1Max) {
        uint equalAmount0;
        uint160 buy0Price;

        //将基金本币换算成 token0
        if(params.amount > 0){
            if(params.token == params.token0){
                equalAmount0 = params.amount0.add(params.amount);
            } else {
                buy0Price = PathPrice.getSqrtPriceX96(buyPath[params.token0],
params.uniV3Factory, true);
                equalAmount0 = params.amount0.add((FullMath.mulDiv(
                    params.amount,
                    FullMath.mulDiv(buy0Price, buy0Price, FixedPoint96.Q96),
                    FixedPoint96.Q96
                )));
            }
        }
        else    equalAmount0 = params.amount0;
```

```
        //将 token1 换算成 token0
        if(params.amount1 > 0){
            equalAmount0 = equalAmount0.add((FullMath.mulDiv(
                params.amount1,
                FixedPoint96.Q96,
                FullMath.mulDiv(params.sqrtPriceX96, params.sqrtPriceX96, FixedPoint96.Q96)
            )));
        }
        require(equalAmount0 > 0, "EIZ");

        // 计算需要的 t0、t1 数量
        (amount0Max, amount1Max) = getAmountsForAmount0(params.sqrtPriceX96, params.sqrtRatioAX96, params.sqrtRatioBX96, equalAmount0);

        // t0 不够，需要补充
        if(amount0Max > params.amount0) {
            //t1 也不够，基金本币需要兑换成 t0 和 t1
            if(amount1Max > params.amount1){
                // 基金本币兑换成 token0
                uint fundToT0;
                if(params.token0 == params.token){
                    fundToT0 = amount0Max - params.amount0;
                    if(fundToT0 > params.amount) fundToT0 = params.amount;
                    amount0Max = params.amount0.add(fundToT0);
                } else {
                    fundToT0 = FullMath.mulDiv(
                        amount0Max - params.amount0,
                        FixedPoint96.Q96,
                        FullMath.mulDiv(buy0Price, buy0Price, FixedPoint96.Q96)
                    );
                    if(fundToT0 > params.amount) fundToT0 = params.amount;
                    if(fundToT0 > 0) {
```

```
                amount0Max =
params.amount0.add(ISwapRouter(params.uniV3Router).exactInput(ISwapRouter.ExactInputParams({
                    path: buyPath[params.token0],
                    recipient: address(this),
                    deadline: block.timestamp,
                    amountIn: fundToT0,
                    amountOutMinimum: 0
                })));
            } else amount0Max = params.amount0;
        }
        // 基金本币兑换成 token1
        if(params.token1 == params.token){
            amount1Max = params.amount1.add(params.amount.sub(fundToT0));
        } else {
            if(fundToT0 < params.amount){
                amount1Max =
params.amount1.add(ISwapRouter(params.uniV3Router).exactInput(ISwapRouter.ExactInputParams({
                    path: buyPath[params.token1],
                    recipient: address(this),
                    deadline: block.timestamp,
                    amountIn: params.amount.sub(fundToT0),
                    amountOutMinimum: 0
                })));
            }
            else amount1Max = params.amount1;
        }
    }
    // t1 多了，多余的 t1 需要兑换成 t0，基金本币全部兑换成 t0
    else {
        // 多余的 t1 兑换成 t0
        if(params.amount1 > amount1Max){
            amount0Max =
params.amount0.add(ISwapRouter(params.uniV3Router).exactInput(ISwapRouter.ExactInputParams({
```

```
                        path: abi.encodePacked(params.token1, params.fee, params.token0),
                        recipient: address(this),
                        deadline: block.timestamp,
                        amountIn: params.amount1.sub(amount1Max),
                        amountOutMinimum: 0
                    }));
                }
                else amount0Max = params.amount0;

                // 基金本币全部兑换成 t0
                if (params.amount > 0){
                    if(params.token0 == params.token){
                        amount0Max = amount0Max.add(params.amount);
                    } else{
                        amount0Max = amount0Max.add(ISwapRouter(params.uniV3Router).exactInput(ISwapRouter.ExactInputParams({
                            path: buyPath[params.token0],
                            recipient: address(this),
                            deadline: block.timestamp,
                            amountIn: params.amount,
                            amountOutMinimum: 0
                        })));
                    }
                }
            }
        }
        // t0 多了，多余的 t0 兑换成 t1，基金本币全部兑换成 t1
        else {
            // 多余的 t0 兑换成 t1
            if(amount0Max < params.amount0){
                amount1Max = params.amount1.add(ISwapRouter(params.uniV3Router).exactInput(ISwapRouter.ExactInputParams({
                    path: abi.encodePacked(params.token0, params.fee, params.token1),
```

```
                    recipient: address(this),
                    deadline: block.timestamp,
                    amountIn: params.amount0.sub(amount0Max),
                    amountOutMinimum: 0
                }));
            }
            else amount1Max = params.amount1;
            // 基金本币全部兑换成 t1
            if(params.amount > 0){
                if(params.token1 == params.token){
                    amount1Max = amount1Max.add(params.amount);
                } else {
                    amount1Max = amount1Max.add(ISwapRouter(params.uniV3Router).exactInput(ISwapRouter.ExactInputParams({
                        path: buyPath[params.token1],
                        recipient: address(this),
                        deadline: block.timestamp,
                        amountIn: params.amount,
                        amountOutMinimum: 0
                    }))); }
            }}}
```

安全建议：无。

3.8. 撤资指定头寸逻辑【通过】

对合约中撤资指定头寸逻辑设计进行审计，检查是否有对参数进行合法性检查，撤资指定头寸逻辑设计是否合理等。

审计结果：经审计，撤资指定头寸逻辑设计合理无误，未发现相关安全风险。

```
/// @inheritdoc IManagerActions
function sub(
    address fund,
```

```
        uint poolIndex,
        uint positionIndex,
        uint proportionX128
    ) external override onlyManager(fund){
        IHotPotV2Fund(fund).sub(poolIndex, positionIndex, proportionX128);
    }
    /// @inheritdoc IHotPotV2FundManagerActions
    function sub(
        uint poolIndex,
        uint positionIndex,
        uint proportionX128
    ) external override onlyController{
        require(poolIndex < pools.length, "IPL");
        require(positionIndex < positions[poolIndex].length, "IPS");

        positions[poolIndex][positionIndex].subLiquidity(Position.SubParams({
            proportionX128: proportionX128,
            pool: pools[poolIndex],
            token: token,
            uniV3Router: uniV3Router
        }), sellPath);
    }
    /// @notice 减少指定头寸 LP，并取回本金本币
    /// @param self 指定头寸
    /// @param params 流动池和要减去的数量
    /// @return amount 获取的基金本币数量
    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();
```

```
            // burn & collect
            (uint amount0, uint amount1) = burnAndCollect(self, params.pool, params.proportionX128);
            //  兑换成基金本币
            if(token0 != params.token && amount0 > 0){
                amount =
ISwapRouter(params.uniV3Router).exactInput(ISwapRouter.ExactInputParams({
                    path: sellPath[token0],
                    recipient: address(this),
                    deadline: block.timestamp,
                    amountIn: amount0,
                    amountOutMinimum: 0
                }));
            }
            //  兑换成基金本币
            if(token1 != params.token && amount1 > 0){
                amount =
amount.add(ISwapRouter(params.uniV3Router).exactInput(ISwapRouter.ExactInputParams({
                    path: sellPath[token1],
                    recipient: address(this),
                    deadline: block.timestamp,
                    amountIn: amount1,
                    amountOutMinimum: 0
                })));
            }
```

安全建议：无。

3.9. 调整头寸投资逻辑【通过】

对合约中调整头寸投资逻辑设计进行审计，检查是否有对参数进行合法性检查，调整头寸投资逻辑设计是否合理等。

审计结果：经审计，调整头寸投资逻辑设计合理无误，未发现安全风险。

```
/// @inheritdoc IManagerActions
function move(
        address fund,
        uint poolIndex,
        uint subIndex,
        uint addIndex,
        uint proportionX128
) external override onlyManager(fund){
        IHotPotV2Fund(fund).move(poolIndex, subIndex, addIndex, proportionX128);
}
/// @inheritdoc IHotPotV2FundManagerActions
function move(
        uint poolIndex,
        uint subIndex,
        uint addIndex,
        uint proportionX128
) external override onlyController {
        require(poolIndex < pools.length, "IPL");
        require(subIndex < positions[poolIndex].length, "ISI");
        require(addIndex < positions[poolIndex].length, "IAI");

        // 移除
        (uint amount0Max, uint amount1Max) = positions[poolIndex][subIndex]
            .burnAndCollect(pools[poolIndex], proportionX128);

        // 添加
        positions[poolIndex][addIndex].addLiquidity(Position.AddParams({
            poolIndex: poolIndex,
            pool: pools[poolIndex],
            amount: 0,
            amount0Max: amount0Max,
            amount1Max: amount1Max,
            token: token,
```

```
            uniV3Router: uniV3Router,
            uniV3Factory: uniV3Factory
        }), sellPath, buyPath);
    }
```

安全建议：无。

3.10. 基金本币存入逻辑【通过】

对合约中用户存入基金本币逻辑设计进行审计，检查是否有对参数进行合法性检查，存入基金本币逻辑设计是否合理等。

审计结果：经审计，未发现存在相关安全风险。

```
    /// @inheritdoc IHotPotV2FundUserActions
    function deposit(uint amount) external override returns(uint share) {
        require(amount > 0, "DAZ");
        uint total_assets = totalAssets();
        TransferHelper.safeTransferFrom(token, msg.sender, address(this), amount);

        return _deposit(amount, total_assets);
    }
    function _deposit(uint amount, uint total_assets) internal returns(uint share) {
        if(totalSupply == 0)
            share = amount;
        else
            share =    FullMath.mulDiv(amount, totalSupply, total_assets);

        investmentOf[msg.sender] = investmentOf[msg.sender].add(amount);
        totalInvestment = totalInvestment.add(amount);
        _mint(msg.sender, share);
        emit Deposit(msg.sender, amount, share);
    }
    /// @inheritdoc IHotPotV2FundState
```

```
function totalAssets() public view override returns (uint amount) {
        amount = IERC20(token).balanceOf(address(this));
        for(uint i = 0; i < pools.length; i++){
            uint _amount;
            (_amount, ) = _assetsOfPool(i);
            amount = amount.add(_amount);
        }
}
  function _assetsOfPool(uint poolIndex) internal view returns (uint amount, uint[] memory) {
        return positions[poolIndex].assetsOfPool(pools[poolIndex], token, sellPath, uniV3Factory);
}
/// @notice 获取某个流动池(pool)，以基金本币衡量的所有资产
/// @param  pool 流动池地址
/// @return amount 资产数量
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.price0 = PathPrice.getSqrtPriceX96(path, uniV3Factory, false);
        }
        if(params.token1 != token){
```

```
            bytes memory path = sellPath[params.token1];
            if(path.length == 0) return(amount, amounts);
            params.price1 = PathPrice.getSqrtPriceX96(path, uniV3Factory, false);
        }

        (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.price0, params.price0, FixedPoint96.Q96),
                    FixedPoint96.Q96);
            }
            else
                _amount = _amount0;

            if(params.token1 != token){
                _amount = _amount.add(FullMath.mulDiv(
                    _amount1,
                    FullMath.mulDiv(params.price1, params.price1, FixedPoint96.Q96),
                    FixedPoint96.Q96));
            }
            else
                _amount = _amount.add(_amount1);

            amounts[i] = _amount;
            amount = amount.add(_amount);
        }
        return(amount, amounts);
    }
    /// @notice 根据设定的兑换路径，获取目标代币的价格平方根
    /// @param path 兑换路径
    /// @param isCurrentPrice 获取当前价格，还是预言机价格
    /// @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,
        bool isCurrentPrice
    ) internal view returns (uint160 sqrtPriceX96){
        require(path.length > 0, "IPL");

        sqrtPriceX96 = uint160(1 << FixedPoint96.RESOLUTION);
```

```
        while (true) {
            (address tokenIn, address tokenOut, uint24 fee) = path.decodeFirstPool();
            IUniswapV3Pool pool = IUniswapV3Pool(PoolAddress.computeAddress(uniV3Factory,
PoolAddress.getPoolKey(tokenIn, tokenOut, fee)));

            uint160 _sqrtPriceX96;
            if(isCurrentPrice){
                (_sqrtPriceX96,,,,,,) = pool.slot0();
            } else {
                uint32[] memory secondAges= new uint32[](2);
                secondAges[0] = 0;
                secondAges[1] = 1;
                (int56[] memory tickCumulatives,) = pool.observe(secondAges);
                _sqrtPriceX96 = TickMath.getSqrtRatioAtTick(int24(tickCumulatives[0] -
tickCumulatives[1]));
            }

            sqrtPriceX96 = uint160(
                tokenIn > tokenOut
                ? FullMath.mulDiv(sqrtPriceX96, FixedPoint96.Q96, _sqrtPriceX96)
                : FullMath.mulDiv(sqrtPriceX96, _sqrtPriceX96, FixedPoint96.Q96)
            );

            // decide whether to continue or terminate
            if (path.hasMultiplePools())
                path = path.skipToken();
            else
                return sqrtPriceX96;
        }
    }
```

安全建议：无。

3.11. 提取指定份额本币【通过】

对合约中的提取指定份额本币逻辑设计安全审计，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。

审计结果：经审计，未发现存在相关安全风险。

```
/// @inheritdoc IHotPotV2FundUserActions
function withdraw(uint share) external override 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
                    }), sellPath);
                    break;
                }
                else {
                    positions[poolIndex][positionIndex].subLiquidity(Position.SubParams({
                        proportionX128: DIVISOR,
                        pool: pools[poolIndex],
                        token: fToken,
                        uniV3Router: uniV3Router
                    }), 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;
        }

        // 处理基金经理分成和基金分成
        if(amount > investment){
```

```
            uint _manager_fee = FullMath.mulDiv(amount.sub(investment), MANAGER_FEE,
DIVISOR);
            uint _fee = FullMath.mulDiv(amount.sub(investment), FEE, DIVISOR);
            TransferHelper.safeTransfer(fToken, manager, _manager_fee);
            TransferHelper.safeTransfer(fToken, controller, _fee);
            amount = amount.sub(_fee).sub(_manager_fee);
        }
        else
            investment = amount;
        // 处理转账
        investmentOf[msg.sender] = investmentOf[msg.sender].sub(investment);
        totalInvestment = totalInvestment.sub(investment);
        _burn(msg.sender, share);

        if(fToken == WETH9){
            IWETH9(WETH9).withdraw(amount);
            TransferHelper.safeTransferETH(msg.sender, amount);
        } else {
            TransferHelper.safeTransfer(fToken, msg.sender, amount);
        }
        emit Withdraw(msg.sender, amount, share);
    }
```

安全建议：无。

3.12. 创建基金逻辑设计【通过】

对合约中的创建基金逻辑设计进行安全审计，检查是否有对参数进行合法性校验，创建基金会逻辑设计是否存在设计缺陷等。

审计结果：经审计，创建基金会逻辑设计合理无误。

```
    /// @inheritdoc IHotPotV2FundFactory
    function createFund(address token, bytes32 descriptor) external override returns (address fund){
```

```
        require(IHotPotV2FundController(controller).verifiedToken(token));
        require(getFund[msg.sender][token] == address(0));

        fund = deploy(WETH9, uniV3Factory, uniV3Router, controller, msg.sender, token, descriptor);
        getFund[msg.sender][token] = fund;

        emit FundCreated(msg.sender, token, fund);
    }
    /// @dev Deploys a fund with the given parameters by transiently setting the parameters storage slot and then
    /// clearing it after deploying the fund.
    /// @param controller The controller address
    /// @param manager The manager address of this fund
    /// @param token The local token address
    /// @param descriptor 32 bytes string descriptor, 8 bytes manager name + 24 bytes brief description
    function deploy(
        address WETH9,
        address uniswapV3Factory,
        address uniswapV3Router,
        address controller,
        address manager,
        address token,
        bytes32 descriptor
    ) internal returns (address fund) {
        parameters = Parameters({
            WETH9: WETH9,
            uniswapV3Factory: uniswapV3Factory,
            uniswapV3Router: uniswapV3Router,
            controller: controller,
            manager: manager,
            token: token,
            descriptor: descriptor
```

```
        });
        fund = address(new HotPotV2Fund{salt: keccak256(abi.encode(manager, token))}());
        delete parameters;
    }
```

安全建议：无。

3.13. receive 逻辑设计【通过】

对合约中的 Receive 逻辑设计进行安全审计，检查是否有对参数进行合法性校验，receive 是否存在设计缺陷等。

审计结果：经审计，未发现存在相关安全风险。

```
    receive() external payable {
        //当前是 WETH9 基金
        if(token == WETH9){
            // 普通用户发起的转账 ETH，认为是 deposit
            if(msg.sender != WETH9 && msg.value > 0){
                uint totals = totalAssets();
                IWETH9(WETH9).deposit{value: address(this).balance}();
                _deposit(msg.value, totals);
            } //else 接收 WETH9 向合约转账 ETH
        }
        // 不是 WETH 基金，不接受 ETH 转账
        else revert();
    }
```

安全建议：无。

3.14. setMaxHarvestSlippage【通过】

对合约中 setMaxHarvestSlippage 的逻辑设计进行安全审计，检查是否有对参数进行合法性校验，setMaxHarvestSlippage 是否存在设计缺陷等。

审计结果：经审计，未发现存在相关安全风险。

```
/// @inheritdoc IGovernanceActions
function setMaxHarvestSlippage(uint slippage) external override onlyGovernance {
    require(slippage <= 100 ,"SMS");
    maxHarvestSlippage = slippage;
    emit SetMaxHarvestSlippage(slippage);
}
```

安全建议：无。

4. 代码基本漏洞检测

4.1. 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 0.5.15 以上，不存在该安全问题。

安全建议：无。

4.2. 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。

4.3. 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。

4.4. 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。

4.5. 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。

4.6. 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。

4.7. 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。

4.8. 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。

4.9. 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。

4.10. 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，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.11. 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。

4.12. 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，智能合约代码中存在增发代币的功能，但由于流动性挖矿需要增发代币，故通过。

安全建议：无。

4.13. 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。

4.14. 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。

4.15. 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、函数、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。

4.16. 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。

4.17. 不安全的接口使用【通过】

检查合约代码实现中是否使用了不安全的接口

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。

4.18. 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。

4.19. 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，智能合约代码不使用结构体，不存在该问题。

安全建议：无。

4.20. 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 Ether，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 Ether 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。

4.21. 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。由于以太坊区块链是公开的，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。

4.22. 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。

4.23. 拒绝服务攻击【通过】

在以太坊的世界中，拒绝服务是致命的，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。

4.24. 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。

4.25. 重入攻击检测【通过】

Solidity 中的 call.value()函数在被用来发送 Ether 的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送 Ether 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。

4.26. 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，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.27. 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。

5. 附录 A：安全风险评级标准

智能合约漏洞评级标准	
漏洞评级	**漏洞评级说明**
高危漏洞	能直接造成代币合约或用户资金损失的漏洞，如：能造成代币价值归零的数值溢出漏洞、能造成交易所损失代币的假充值漏洞、能造成合约账户损失 ETH 或代币的重入漏洞等； 能造成代币合约归属权丢失的漏洞，如：关键函数的访问控制缺陷、call 注入导致关键函数访问控制绕过等； 能造成代币合约无法正常工作的漏洞，如：因向恶意地址发送 ETH 导致的拒绝服务漏洞、因 energy 耗尽导致的拒绝服务漏洞。
中危漏洞	需要特定地址才能触发的高风险漏洞，如代币合约拥有者才能触发的数值溢出漏洞等；非关键函数的访问控制缺陷、不能造成直接资金损失的逻辑设计缺陷等。
低危漏洞	难以被触发的漏洞、触发之后危害有限的漏洞，如需要大量 ETH 或代币才能触发的数值溢出漏洞、触发数值溢出后攻击者无法直接获利的漏洞、通过指定高 energy 触发的事务顺序依赖风险等。

6. 附录 B：智能合约安全审计工具简介

6.1 Manticore

Manticore 是一个分析二进制文件和智能合约的符号执行工具，Manticore 包含一个符号虚拟机（EVM），一个 EVM 反汇编器/汇编器以及一个用于自动编译和分析 Solidity 的方便界面。它还集成了 Ethersplay，用于 EVM 字节码的 Bit of Traits of Bits 可视化反汇编程序，用于可视化分析。与二进制文件一样，Manticore 提供了一个简单的命令行界面和一个用于分析 EVM 字节码的 Python API。

6.2 Oyente

Oyente 是一个智能合约分析工具，Oyente 可以用来检测智能合约中常见的 bug，比如 reentrancy、事务排序依赖等等。更方便的是，Oyente 的设计是模块化的，所以这让高级用户可以实现并插入他们自己的检测逻辑，以检查他们的合约中自定义的属性。

6.3 securify.sh

Securify 可以验证智能合约常见的安全问题，例如交易乱序和缺少输入验证，它在全自动化的同时分析程序所有可能的执行路径，此外，Securify 还具有用于指定漏洞的特定语言，这使 Securify 能够随时关注当前的安全性和其他可靠性问题。

6.4 Echidna

Echidna 是一个为了对 EVM 代码进行模糊测试而设计的 Haskell 库。

6.5 MAIAN

MAIAN 是一个用于查找智能合约漏洞的自动化工具，Maian 处理合约的字

节码，并尝试建立一系列交易以找出并确认错误。

6.6 ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

6.7 ida-evm

ida-evm 是一个针对虚拟机（EVM）的 IDA 处理器模块。

6.8 Remix-ide

Remix 是一款基于浏览器的编译器和 IDE，可让用户使用 Solidity 语言构建合约并调试交易。

6.9 知道创宇区块链安全审计人员专用工具包

知道创宇安全审计人员专用工具包，由知道创宇渗透测试工程师研发，收集和使用，包含专用于测试人员的批量自动测试工具，自主研发的工具、脚本或利用工具等。

知道创宇

北京知道创宇信息技术股份有限公司

咨询电话 +86(10)400 060 9587

邮　　箱 sec@knownsec.com

官　　网 www.knownsec.com

地　　址 北京市 朝阳区 望京 SOHO T2-B座-2509