Version Summary

Content	Date	Version
Editing Document	2021/08/31	V1.1

Report Information

Title	Version	Document Number	Type
DeepGo NudgePool Smart Contract Audit Report	V1.0		Open to project team

Copyright Notice

Table of Contents

1. Introduction

The effective test time of this report is from **August 3**, **2021** to **August 31**, **2021** . During this period, the security and standardization of **the smart contract code of the DeepGo NudgePool** will be audited and used 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 4). **The smart contract code of the DeepGo NudgePool** is comprehensively assessed as

Results of this smart contract security audit：

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.

Report information of this audit:

Report Number：

Report query address link:

Target information of the DeepGo NudgePool audit:

Target information	
Project name	DeepGo NudgePool
Token address	

Code type	Polygon / BSC / Eth smart contract code
Code language	Solidity

Contract documents and hash:

Contract documents	MD5
NPProxy.sol	A9F40E9BF2281F721B77E725C99271A2
NudgePool.sol	CF4598498A5487D269678FC562C0C971
NudgePoolStatus.sol	4FD9E94CFC0036283BD43CDD9EC1B57F
GPStorage.sol	2DC3BA1AEBE7BE2A5E08A94580781EBA
IPStorage.sol	B8E8903C7093E171E90970E4EB3156E7
LPStorage.sol	E1ED3714C0852447225CF8D5E92B2BAA
NPStorage.sol	A09B3E81A0E741101B607B2A02AFC32D
VaultStorage.sol	80B1AFCC7D12FBB180F728794456D250
BaseLogic.sol	A3D824C67AE4A896B1B0B9CD9A0AA54A
GPDepositLogic.so	A3A6B9F326B50192754A6C94B7F5998F
GPWithdrawLogic.sol	4C7D36E54B717DE49BDE31797929D091
IPLogic.sol	843E9FA6F7A402C9ED472597A47F03D4
LiquidationLogic.sol	86FBBF7A143675F54535F01AE7F2948C
LPLogic.sol	5AE67CA8375DB776ED079A2A08EB53B6
StateLogic.sol	66371E13D85A3EEAE4C1BCBBF1E14733
VaultLogic.sol	2233F67DBFA099048C351DFA2DDB9ABE

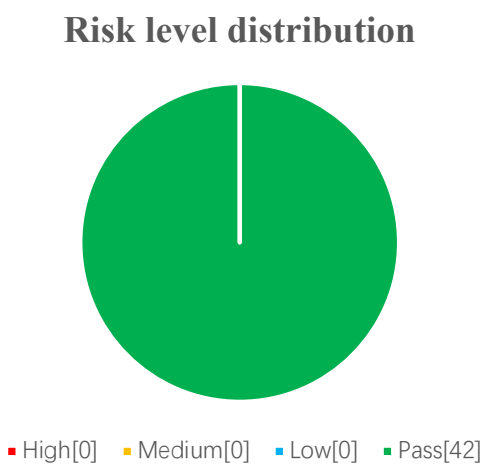
BytesUtils.sol	A4E84BF39B6E1B6772274BB5DA63668A
NPSwap.sol	37F089AF82AC5CF0EECA455656EF5424
Ownable.sol	D656B64B74C791BE0C68D6DF0F0A0CA8
SafeMath.sol	C4EA4B909BA18A48E19FFB696704B94F

2. Code vulnerability analysis

2.1 Vulnerability Level Distribution

Vulnerability risk statistics by level：

Vulnerability risk level statistics table			
High	Medium	Low	Pass
0	0	0	42



2.2 Audit Result

Result of audit			
Audit Target	**Audit**	**Status**	**Audit Description**
Business security testing	**initialize logic design**	Pass	After testing, there is no such safety vulnerability.
	createPool logic design	Pass	After testing, there is no such safety vulnerability.
	auctionPool logic design	Pass	After testing, there is no such safety vulnerability.
	changePoolParam logic design	Pass	After testing, there is no such safety vulnerability.
	IPDepositRunning logic design	Pass	After testing, there is no such safety vulnerability.
	LPDepositRaising logic design	Pass	After testing, there is no such safety vulnerability.
	LPDepositRunning logic design	Pass	After testing, there is no such safety vulnerability.
	LPDoDepositRunning logic design	Pass	After testing, there is no such safety vulnerability.
	LPWithdrawRunning logic design	Pass	After testing, there is no such safety vulnerability.
	withdrawVault logic design	Pass	After testing, there is no such safety vulnerability.
	GPDepositRaising logic design	Pass	After testing, there is no such safety vulnerability.

	GPDepositRunning logic design	Pass	After testing, there is no such safety vulnerability.
	GPDoDepositRunning logic design	Pass	After testing, there is no such safety vulnerability.
	GPWithdrawRunning logic design	Pass	After testing, there is no such safety vulnerability.
	computeVaultReward logic design	Pass	After testing, there is no such safety vulnerability.
Basic code vulnerability detection	**Compiler version security**	Pass	After testing, there is no such safety vulnerability.
	Redundant code	Pass	After testing, there is no such safety vulnerability.
	Use of safe arithmetic library	Pass	After testing, there is no such safety vulnerability.
	Not recommended encoding	Pass	After testing, there is no such safety vulnerability.
	Reasonable use of require/assert	Pass	After testing, there is no such safety vulnerability.
	fallback function safety	Pass	After testing, there is no such safety vulnerability.
	tx.origin authentication	Pass	After testing, there is no such safety vulnerability.
	Owner permission control	Pass	After testing, there is no such safety vulnerability.
	Gas consumption detection	Pass	After testing, there is no such safety vulnerability.
	call injection attack	Pass	After testing, there is no such safety vulnerability.

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

	Fake recharge vulnerability detection	Pass	After testing, there is no such safety vulnerability.
	Reentry attack detection	Pass	After testing, there is no such safety vulnerability.
	Replay attack detection	Pass	After testing, there is no such safety vulnerability.
	Rearrangement attack detection	Pass	After testing, there is no such safety vulnerability.

3. Analysis of code audit results

3.1. initialize logic design【PASS】

Perform security audits on the initialize logic design in the contract to check whether there are logic design flaws, whether it can be initialized multiple times, etc.

Audit analysis: initialize initialization logic design is reasonable and correct, and no related security risks are found.

```
        function initialize(
            address _ipc,
            address _gpdc,
            address _gpwc,
            address _lpc,
            address _vtc,
            address _stc,
            address _lqdc
        )
            external onlyOwner
        {
            require(!initialized, "Already Initialized");
            setUpgrade("0.0.1", _ipc, _gpdc, _gpwc, _lpc, _vtc, _stc, _lqdc);
            executeUpgrade();
            initialized = true;
        }
    function setUpgrade(
        string memory _newVersion,
        address _ipc,
        address _gpdc,
        address _gpwc,
        address _lpc,
        address _vtc,
        address _stc,
        address _lqdc
```

```
    )
        public onlyOwner
    {
        require(_ipc != address(0) && _gpdc != address(0) && _gpwc != address(0) &&
                _lpc != address(0) && _vtc != address(0) && _stc != address(0) &&
                _lqdc != address(0), "Wrong Address");
        require(bytes(_newVersion).length > 0, "Empty Version");
        delayVersionName = _newVersion;
        delayVersion.ipc = _ipc;
        delayVersion.gpdc = _gpdc;
        delayVersion.gpwc = _gpwc;
        delayVersion.lpc = _lpc;
        delayVersion.vtc = _vtc;
        delayVersion.stc = _stc;
        delayVersion.lqdc = _lqdc;
        startTime = block.timestamp;
        emit SetUpgrade(_newVersion, _ipc, _gpdc, _gpwc, _lpc, _vtc, _stc, _lqdc);
    }
    function executeUpgrade(
    )
        public onlyOwner
    {
        require(delayVersion.ipc != address(0) && delayVersion.gpdc != address(0) &&
delayVersion.gpwc != address(0) &&
                delayVersion.lpc != address(0) && delayVersion.vtc != address(0) &&
delayVersion.stc != address(0) &&
                delayVersion.lqdc != address(0), "Wrong Address");
        if (initialized == true) {
            require(block.timestamp > startTime.add(delayTime), "In Delay" );
        }
        versions[delayVersionName] = delayVersion;
        versionName = delayVersionName;
        curVersion = delayVersion;
```

```
            versionList.push(delayVersionName);
            delayVersionName = '';
            delete delayVersion;
            emit ExecuteUpgrade(versionName, curVersion.ipc, curVersion.gpdc, curVersion.gpwc,
curVersion.lpc,
            curVersion.vtc, curVersion.stc, curVersion.lqdc);
    }
```

Recommendation： nothing.

3.2. createPool logic design【PASS】

Perform security audits on the logic design of the creators (IPs) in the contract to establish the token pool, check whether there is a check on the parameters, whether there is a risk of plastic overflow, etc.

Audit analysis: The logical design of the establishment of the token pool is reasonable and correct, and no related security risks have been found.

```
    //  Establish a token pool
    function createPool(
            address _ip,            //  The address of the token pool holder
            address _ipToken,       //  iptoken address
            address _baseToken,     //  basetoken address
            uint256 _ipTokensAmount,    //  Number of assets in IPtoken
            uint256 _dgtTokensAmount,   //  Number of assets in DGTtoken
            uint32 _ipImpawnRatio,      //  Mortgage rate
            uint32 _ipCloseLine,        //  Liquidation line
            uint32 _chargeRatio,        //  Commission rate
            uint256 _duration           //  Auction duration
        )
            external whenNotPaused      //  Request that this contract is not suspended
        {
            (bool status,) = curVersion.ipc.delegatecall(abi.encodeWithSelector(bytes4(keccak256(
```

```
"createPool(address,address,address,uint256,uint256,uint32,uint32,uint32,uint256)")),
                    _ip, _ipToken, _baseToken, _ipTokensAmount, _dgtTokensAmount,
                    _ipImpawnRatio, _ipCloseLine, _chargeRatio, _duration));
                // Call the createPool function of LPLogic.sol to create a token pool
                require(status == true, "Create Pool Failed"); // Determine whether the establishment is
successful
                emit CreatePool(_ip, _ipToken, _baseToken, _ipTokensAmount, _dgtTokensAmount,
                                    _ipImpawnRatio, _ipCloseLine, _chargeRatio, _duration);
            }// Record token pool establishment events
        function createPool(
                address _ip,
                address _ipToken,
                address _baseToken,
                uint256 _ipTokensAmount,
                uint256 _dgtTokensAmount,
                uint32 _ipImpawnRatio,
                uint32 _ipCloseLine,
                uint32 _chargeRatio,
                uint256 _duration
            )
                external
                poolNotExist(_ipToken, _baseToken)     // Determine whether the token pair already
exists
            {

                qualifiedIP(_ip, _ipToken, _baseToken, _ipTokensAmount, _dgtTokensAmount, true);
               // Call the qualifiedIP function to check whether this IP is qualified
                checkIPParams(_ipImpawnRatio, _ipCloseLine, _chargeRatio, _duration);
               // Call the checkIPParams function to check whether the IP parameters meet the
conditions
                IERC20(_ipToken).safeTransferFrom(_ip, address(this), _ipTokensAmount);
               // Transfer _iptoken funds to this contract
```

```
                IERC20(DGTToken).safeTransferFrom(_ip, address(this), _dgtTokensAmount);
            //  Transfer DGTToken funds to this contract
                _IPS.insertPool(_ipToken, _baseToken); //  Call the insertPool function to set the token pool parameters
                _IPS.setPoolStage(_ipToken, _baseToken, uint8(Stages.CREATING));
            //  Set the status of the token pool to being created
                _IPS.setPoolAuctionEndTime(_ipToken, _baseToken, block.timestamp.add(auctionDuration)); //  Set the auction duration to seven days

                _IPS.setIPAddress(_ipToken, _baseToken, _ip); //  Set the address of the holder of the token pool
                _IPS.setIPTokensAmount(_ipToken, _baseToken, _ipTokensAmount);
            //  Set the number of holders of the token pool_iptoken
                _IPS.setDGTTokensAmount(_ipToken, _baseToken, _dgtTokensAmount);
            //  Set the number of DGT tokens of the holder of the token pool
                _IPS.setIPImpawnRatio(_ipToken, _baseToken, _ipImpawnRatio);
            //  Set the mortgage rate of the token pool
                _IPS.setIPCloseLine(_ipToken, _baseToken, _ipCloseLine);
            //  Set up the liquidation line of the token pool
                _IPS.setIPChargeRatio(_ipToken, _baseToken, _chargeRatio);
            //  Set the commission rate of the token pool
                _IPS.setIPDuration(_ipToken, _baseToken, _duration);
            //  Set the duration of the token pool holder
                poolTransitNextStage(_ipToken, _baseToken);
            //  The token pool enters the next state, the auction state
            }
    function insertPool(address _ipt, address _bst) external {
            8(proxy == msg.sender, "Not Permit");
            require(pools[_ipt][_bst].valid == false, "Pool Already Exist");

            poolsArray.push(Pool(_ipt, _bst));
            pools[_ipt][_bst].valid = true;
            pools[_ipt][_bst].locked = false;
```

```
        pools[_ipt][_bst].id = poolsArray.length;
        pools[_ipt][_bst].createdTime = block.timestamp;
    }
    function setPoolStage(address _ipt, address _bst, uint8 _stage) external {
        require(proxy == msg.sender, "Not Permit");
        pools[_ipt][_bst].stage = _stage;
    }
    function setPoolAuctionEndTime(address _ipt, address _bst, uint256 _time) external {
        require(proxy == msg.sender, "Not Permit");
        pools[_ipt][_bst].auctionEndTime = _time;
    }
    function setIPAddress(address _ipt, address _bst, address _ip) external {
        require(proxy == msg.sender, "Not Permit");
        pools[_ipt][_bst].IP.ip = _ip;
    }
    function setDGTTokensAmount(address _ipt, address _bst, uint256 _amount) external {
        require(proxy == msg.sender, "Not Permit");
        pools[_ipt][_bst].IP.dgtTokensAmount = _amount;
    }
    function poolTransitNextStage(
        address _ipToken,
        address _baseToken
    )
        internal
    {
        uint8 stage = _IPS.getPoolStage(_ipToken, _baseToken);
        Stages next = Stages(stage + 1);
        uint8 nexStage = uint8(next);

        require(nexStage > stage, "Wrong Stage Transit");
        _IPS.setPoolStage(_ipToken, _baseToken, nexStage);
    }
    function isContract(address account) internal view returns (bool) {
```

```
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            size := extcodesize(account)
        }
        return size > 0;
    }
```

Recommendation： nothing.

3.3. auctionPool logic design【PASS】

Perform security audits on the logic design of the GPs auction token pool in the contract to check whether there is an integer overflow, a denial of service or a logic design error.

Audit analysis: The logic design of the auction token pool is reasonable and correct, and no related security risks have been found.

```
function auctionPool(
        address _ip,
        address _ipToken,
        address _baseToken,
        uint256 _ipTokensAmount,
        uint256 _dgtTokensAmount
    )
        external whenNotPaused
    {
        (bool status,) = curVersion.ipc.delegatecall(abi.encodeWithSelector(bytes4(keccak256(
            "auctionPool(address,address,address,uint256,uint256)")),
            _ip, _ipToken, _baseToken, _ipTokensAmount, _dgtTokensAmount));
```

```
        // Call the auctionPool function of LPLogic.sol to participate in the auction
        require(status == true, "Auction Pool Failed");  // Judge whether it is successful
        emit AuctionPool(_ip, _ipToken, _baseToken, _ipTokensAmount, _dgtTokensAmount);
    }
    // Record auction events
    function auctionPool(
            address _ip,
            address _ipToken,
            address _baseToken,
            uint256 _ipTokensAmount,
            uint256 _dgtTokensAmount
        )
            external
            lockPool(_ipToken, _baseToken)    // Lock the token pool
        {
            poolAtStage(_ipToken, _baseToken, Stages.AUCTING);// Check whether the token
pool status is in auction
            qualifiedIP(_ip, _ipToken, _baseToken, _ipTokensAmount, _dgtTokensAmount, false);
              // Check whether the parameters are qualified and whether the number of dgtTokens
is greater than 105% of _ipTokens
            require(block.timestamp < _IPS.getPoolAuctionEndTime(_ipToken, _baseToken),
"Auction End");
            // Determine whether the auction is over, and you cannot participate when it is over
            IERC20(_ipToken).safeTransferFrom(_ip, address(this), _ipTokensAmount);
            // Transfer _ipToken funds to this contract
            IERC20(DGTToken).safeTransferFrom(_ip, address(this), _dgtTokensAmount);
            // Transfer DGTToken funds to this contract
            address oriIP = _IPS.getIPAddress(_ipToken, _baseToken);
              // Get the address of the current holder of the token pool

IERC20(_ipToken).safeTransfer(oriIP,_IPS.getIPTokensAmount(_ipToken,_baseToken));
              // Return the _ipToken funds of the current holder
```

```
IERC20(DGTToken).safeTransfer(oriIP,_IPS.getDGTTokensAmount(_ipToken,_baseToken));
                    // Return the current holder's DGTToken funds
                _IPS.setIPAddress(_ipToken, _baseToken, _ip);
                    // Change the address of the token pool holder
                _IPS.setIPTokensAmount(_ipToken, _baseToken, _ipTokensAmount);
                    // Set the holder's _ipToken number
                _IPS.setDGTTokensAmount(_ipToken, _baseToken, _dgtTokensAmount);
                    // Set the number of DGTToken holders
        if (auctionDuration > 30 minutes &&
            block.timestamp > _IPS.getPoolAuctionEndTime(_ipToken, _baseToken).sub(30 minutes))
{
                _IPS.setPoolAuctionEndTime(_ipToken, _baseToken, block.timestamp.add(30 minutes));
        }
          // Once someone participates in the auction, set the remaining auction time to 30 minutes
          }
```

Recommendation： nothing.

3.4. changePoolParam logic design【PASS】

Perform security audits on the logic design of changing token pool parameters in the contract, and check whether there is a check on the parameters and whether there are logic design defects, etc.

Audit analysis: The logic design of changing the token pool parameters is reasonable and correct, and no related security risks have been found.

```
function changePoolParam(
            address _ipToken,
            address _baseToken,
            uint32 _ipImpawnRatio,
            uint32 _ipCloseLine,
            uint32 _chargeRatio,
            uint256 _duration
```

```
    )
        external
        lockPool(_ipToken, _baseToken)          // Lock the token pool during the change
    {
        address ip = _IPS.getIPAddress(_ipToken, _baseToken);   // Get the address of the holder
of the token pool
        require(msg.sender == ip, "Not Permit");        // Non-holder cannot change
        poolAtStage(_ipToken, _baseToken, Stages.AUCTING);
        // Check the status of the token pool
        checkIPParams(_ipImpawnRatio, _ipCloseLine, _chargeRatio, _duration);
        // Check whether the parameters are reasonable
        _IPS.setIPImpawnRatio(_ipToken, _baseToken, _ipImpawnRatio);
        _IPS.setIPCloseLine(_ipToken, _baseToken, _ipCloseLine);
        _IPS.setIPChargeRatio(_ipToken, _baseToken, _chargeRatio);
        _IPS.setIPDuration(_ipToken, _baseToken, _duration);
        // Change parameters
    }
```

Recommendation： nothing.

3.5. IPDepositRunning logic design【PASS】

Perform security audits on the additional investment logic design of token pool holders in the contract to check whether there is an integer overflow or logic design error.

Audit analysis: The additional investment logic design is reasonable and correct, and no related security risks are found.

```
    function IPDepositRunning(
            address _ipToken,   //_ipToken address
            address _baseToken,   //_baseToken address
            uint256 _ipTokensAmount   //_ipToken quantity
        )
            external whenNotPaused
```

```
        returns (uint256 amount)
    {
        (bool status, bytes memory data) = curVersion.ipc.delegatecall(
            abi.encodeWithSelector(bytes4(keccak256(
            "IPDepositRunning(address,address,uint256)")),
            _ipToken, _baseToken, _ipTokensAmount));
        // Call the IPDepositRunning function of LPLogic.sol
        require(status == true, "IP Deposit Failed");
        // Check if the call is successful
        amount = data.bytesToUint256();
        emit RunningIPDeposit(_ipToken, _baseToken, _ipTokensAmount);
        // Log event
        return amount;
    }
function IPDepositRunning(
        address _ipToken,
        address _baseToken,
        uint256 _ipTokensAmount
    )
        external
        lockPool(_ipToken, _baseToken)      // Lock the token pool
        returns (uint256 amount)
    {
        address ip = _IPS.getIPAddress(_ipToken, _baseToken);// Get the address of the token
pool owner
        require(ip == msg.sender, "Not Permit");// Non-token owners cannot control
        amount = _ipTokensAmount;
        IERC20(_ipToken).safeTransferFrom(ip, address(this), amount);
        // Transfer _ipToken funds to this contract
        uint256 curAmount = _IPS.getIPTokensAmount(_ipToken, _baseToken);
        // Get the number of _ipToken of the token pool holder
        _IPS.setIPTokensAmount(_ipToken, _baseToken, curAmount.add(amount));
        // Update the number of _ipToken of token pool holders
```

```
        return amount;
    }
```

Recommendation： nothing.

3.6. LPDepositRaising logic design【PASS】

Perform security audits on the logic design of the LPs in the contract investing funds during the token pool RAISING period to check whether there is an integer overflow or a logic design error.

Audit analysis: The logic design of the invested funds is reasonable and correct, and no related security risks are found.

```
function LPDepositRaising(
        address _ipToken,
        address _baseToken,
        uint256 _baseTokensAmount,
        bool _create
)
        external
        lockPool(_ipToken, _baseToken)
        returns (uint256 amount)
{
        poolAtStage(_ipToken, _baseToken, Stages.RAISING);
        address _lp = msg.sender;
        uint256 oriLPAmount = _LPS.getCurLPAmount(_ipToken, _baseToken);

        amount = _baseTokensAmount;
        IERC20(_baseToken).safeTransferFrom(_lp, address(this), amount);
        _LPS.setCurLPAmount(_ipToken, _baseToken, oriLPAmount.add(amount));
        if (_create) {
            _LPS.insertLP(_ipToken, _baseToken, _lp, amount, false);
        } else {
            uint256 oriAmount = _LPS.getLPBaseAmount(_ipToken, _baseToken, _lp);
```

```
            _LPS.setLPBaseAmount(_ipToken, _baseToken, _lp, oriAmount.add(amount));
        }

        require(amount > 0, "Deposit Zero");
        return amount;
    }
```

Recommendation： nothing.

3.7. LPDepositRunning logic design【PASS】

Perform security audits on the logic design of the LPs in the contract investing funds during the token pool RUNNING period to check whether there is an integer overflow or a logic design error.

Audit analysis: The logic design of the invested funds is reasonable and correct, and no related security risks are found.

```
    function LPDepositRunning(
        address _ipToken,
        address _baseToken,
        uint256 _baseTokensAmount,
        bool _create
    )
        external
        lockPool(_ipToken, _baseToken)
        returns (uint256 amount)
    {
        poolAtStage(_ipToken, _baseToken, Stages.RUNNING);
        address _lp = msg.sender;

        amount = _baseTokensAmount;
        IERC20(_baseToken).safeTransferFrom(_lp, address(this), amount);
        if (_create) {
            _LPS.insertLP(_ipToken, _baseToken, _lp, amount, true);
```

```
        } else {
            uint256 oriAmount = _LPS.getLPRunningDepositAmount(_ipToken, _baseToken, _lp);
            require(oriAmount == 0, "Wait To Do");
            _LPS.setLPRunningDepositAmount(_ipToken, _baseToken, _lp, amount);
        }
        require(amount > 0, "Deposit Zero");
        return amount;
    }
```

Recommendation： nothing.

3.8. LPDoDepositRunning logic design【PASS】

Perform security audits on the LPs in the contract to confirm the input logic design to check whether there is an integer overflow or a logic design error.

Audit analysis: confirmed that the input logic design is reasonable and correct, and no related security risks are found.

```
    function LPDoDepositRunning(
        address _ipToken,   //  _ipToken address
        address _baseToken, //  _baseToken address
    )
        external whenNotPaused
    {
        (bool status,) = curVersion.lpc.delegatecall(
            abi.encodeWithSelector(bytes4(keccak256(
            "LPDoDepositRunning(address,address)")), _ipToken, _baseToken));
        ·// Call the LPDoDepositRunning function of LPLogic.sol
        require(status == true, "LP Do Deposit Failed");
          // Determine whether the call is successful
        emit RunningLPDoDeposit(_ipToken, _baseToken);
          // Log event
    }
    function LPDoDepositRunning(
```

```
        address _ipToken,
        address _baseToken
    )
        external
        lockPool(_ipToken, _baseToken)      //  Lock the token pool
    {
        poolAtStage(_ipToken, _baseToken, Stages.RUNNING);         //  Determine the status
of the token pool
        address _lp = msg.sender;
        uint256 oriLPAmount = _LPS.getCurLPAmount(_ipToken, _baseToken);
       //  Get the total amount of funds in the token pool LPs
        uint256 oriAmount = _LPS.getLPBaseAmount(_ipToken, _baseToken, _lp);
       //  Get the amount of funds in the account
        uint256 curRaiseLP = _GPS.getCurRaiseLPAmount(_ipToken, _baseToken);
       //  Get the amount of funds raised in the token pool
        uint256 amount = _LPS.getLPRunningDepositAmount(_ipToken, _baseToken, _lp);
       //  Get the amount of funds invested when the account is running
        require(amount > 0, "Already done");
        _LPS.setLPRunningDepositAmount(_ipToken, _baseToken, _lp, 0);
          //  Clear settings
        _LPS.setCurLPAmount(_ipToken, _baseToken, oriLPAmount.add(amount));
       //  Update the total amount of funds of all LPs in the token pool
        _LPS.setLPBaseAmount(_ipToken, _baseToken, _lp, oriAmount.add(amount));
       //  Update the amount of funds in the current account of the token pool
        lendToGP(_ipToken, _baseToken, oriLPAmount.add(amount).sub(curRaiseLP));
       //  Update GP loan amount
    }
    function lendToGP(
        address _ipToken,
        address _baseToken,
        uint256 _amount
    )
        private
```

```
        returns (uint256 lend)
    {
        uint256 inUnit = 10**ERC20(_ipToken).decimals();
        uint256 price = NPSwap.getAmountOut(_ipToken, _baseToken, inUnit);
        uint256 IPAmount = _GPS.getCurIPAmount(_ipToken, _baseToken);
        uint256 raiseLP = _GPS.getCurRaiseLPAmount(_ipToken, _baseToken);
        uint256 balance = IPAmount.mul(price).div(inUnit);

        if (raiseLP >= balance ||
            balance.mul(RATIO_FACTOR).div(balance.sub(raiseLP)) >=
            raiseRatio + RATIO_FACTOR) {
            lend = 0;
            return lend;
        }

        uint256 GPBalance = balance.sub(raiseLP);
        lend = GPBalance.mul(raiseRatio +
RATIO_FACTOR).div(RATIO_FACTOR).sub(balance);
        lend = lend > _amount ? _amount : lend;

        uint256 swappedIP = NPSwap.swap(_baseToken, _ipToken, lend);
        _GPS.setCurRaiseLPAmount(_ipToken, _baseToken, raiseLP.add(lend));
        _GPS.setCurIPAmount(_ipToken, _baseToken, IPAmount.add(swappedIP));
        allocateFunds(_ipToken, _baseToken);

        return lend;
    }

    function allocateFunds(
        address _ipToken,
        address _baseToken
    )
        private
```

```
{
    uint256 len = _GPS.getGPArrayLength(_ipToken, _baseToken);
    uint256 balance = _GPS.getCurGPBalance(_ipToken, _baseToken);
    uint256 IPAmount = _GPS.getCurIPAmount(_ipToken, _baseToken);
    uint256 raiseLP = _GPS.getCurRaiseLPAmount(_ipToken, _baseToken);
    uint256 resIPAmount = IPAmount;
    uint256 resRaiseLP = raiseLP;

    for (uint256 i = 0; i < len; i++) {
        address gp = _GPS.getGPByIndex(_ipToken, _baseToken, i);
        uint256 gpBalance = _GPS.getGPBaseBalance(_ipToken, _baseToken, gp);
        uint256 curAmount = gpBalance.mul(IPAmount).div(balance);
        resIPAmount -= curAmount;
        curAmount = i == len - 1 ? curAmount.add(resIPAmount) : curAmount;
        _GPS.setGPHoldIPAmount(_ipToken, _baseToken, gp, curAmount);

        curAmount = gpBalance.mul(raiseLP).div(balance);
        resRaiseLP -= curAmount;
        curAmount = i == len - 1 ? curAmount.add(resRaiseLP) : curAmount;
        _GPS.setGPRaiseLPAmount(_ipToken, _baseToken, gp, curAmount);
    }
}
```

Recommendation： nothing.

3.9. LPWithdrawRunning logic design【PASS】

Perform security audits on the logic design of LPs withdrawal funds in the contract to check whether there is an integer overflow, reentry attack or logic design error.

Audit analysis: The logic design of the withdrawal of funds is reasonable and correct, and no related security risks are found.

```
function LPWithdrawRunning(
```

```
        address _ipToken,     //  _ipToken address
        address _baseToken, //  _baseToken address
        uint256 _baseTokensAmount,   //  _baseTokens number
         bool _vaultOnly
        )
            external whenNotPaused
            returns (uint256 amount)
        {
            (bool status, bytes memory data) = curVersion.lpc.delegatecall(
                abi.encodeWithSelector(bytes4(keccak256(
                "LPWithdrawRunning(address,address,uint256,bool)")),
                _ipToken, _baseToken, _baseTokensAmount, _vaultOnly));
            require(status == true, "LP Withdraw Failed");
            amount = data.bytesToUint256();
            emit RunningLPWithdraw(_ipToken, _baseToken, _baseTokensAmount);
            return amount;
        }
    function LPWithdrawRunning(
            address _ipToken,
            address _baseToken,
            uint256 _baseTokensAmount,
            bool _vaultOnly
        )
            external
            lockPool(_ipToken, _baseToken)              //  Lock the token pool
            returns (uint256 amount)
        {
            poolAtStage(_ipToken, _baseToken, Stages.RUNNING); //  Determine the status of the
token pool
            address _lp = msg.sender;

            if (_vaultOnly) {
                amount = _LPS.getLPVaultReward(_ipToken, _baseToken, _lp);
```

```
            // Get reward amount
            _LPS.setLPVaultReward(_ipToken, _baseToken, _lp, 0);
            // Cleared
            IERC20(_baseToken).safeTransfer(_lp, amount);
            // Send reward amount
            return amount;
        }
        // Withdraw all base token, ignore input amount
        _baseTokensAmount = _LPS.getLPBaseAmount(_ipToken, _baseToken, _lp);
        // Get the amount of fixed funds in the account
        amount = reclaimFromGP(_ipToken, _baseToken, _baseTokensAmount);
        // Recover funds from GP
        amount = amount.add(_LPS.getLPVaultReward(_ipToken, _baseToken, _lp));
    // Get reward amount
        amount = amount.sub(chargeFee(_ipToken, _baseToken, _lp));
    // Deduction rate
        IERC20(_baseToken).safeTransfer(_lp, amount);
    // Send the remaining reward funds
        _LPS.deleteLP(_ipToken, _baseToken, _lp);
    // Delete account
        return amount;
    }
```

Recommendation： nothing.

3.10. withdrawVault logic design【PASS】

Perform security audits on the logic design of IPs withdrawal funds in the contract to check whether there is an integer overflow, reentry attack or logic design error.

Audit analysis: The logical design of the withdrawal of funds is reasonable and correct, and no relevant security risks are found.

```
function withdrawVault(
```

```
        address _ipToken,     //  _ipToken address
        address _baseToken, //  _baseToken address
        uint256 _baseTokensAmount    //  _baseTokens number


        )
            external whenNotPaused
            returns (uint256 amount)
        {
            (bool status, bytes memory data) = curVersion.vtc.delegatecall(
                abi.encodeWithSelector(bytes4(keccak256(
                "withdrawVault(address,address,uint256)")),
                _ipToken, _baseToken, _baseTokensAmount));
          //  Call the withdrawVault function of VaultLogic.sol
            require(status == true, "Withdraw Vault Failed");
           //  Determine whether the call is successful
            amount = data.bytesToUint256();
            emit WithdrawVault(_ipToken, _baseToken, _baseTokensAmount);
           //  Log event
            return amount;
        }
    function withdrawVault(
            address _ipToken,
            address _baseToken,
            uint256 _baseTokensAmount
        )
            external
            returns (uint256 amount)
        {
            // IP can only withdraw 80% money from total vault
            poolAtStage(_ipToken, _baseToken, Stages.RUNNING);         //  Determine the status
of the token pool
            address ip = _IPS.getIPAddress(_ipToken, _baseToken);
            //  Get the address of the token pool holder
```

```
            uint256 vault = _VTS.getTotalVault(_ipToken, _baseToken);
            // Get all the funds in the token pool
            uint256 withdrawed = _VTS.getIPWithdrawed(_ipToken, _baseToken);
            // Get the funds that the owner of the token pool has withdrawn
            uint256 curVault = _VTS.getCurVault(_ipToken, _baseToken);
            // Get current funds in the token pool
            vault = vault.mul(80).div(100);
            require(msg.sender == ip, "Not Permit");
            // Token pool holders can withdraw
            require(vault.sub(withdrawed) >= _baseTokensAmount, "Withdraw too much");
            // The number of funds withdrawn does not exceed the number of funds that can be
withdrawn
            amount = curVault > _baseTokensAmount ? _baseTokensAmount : curVault;
            // The number of withdrawals does not exceed the total number of _baseToken
            IERC20(_baseToken).safeTransfer(ip, amount);
            // Send withdrawal funds
            curVault = curVault.sub(amount);
            _VTS.setCurVault(_ipToken, _baseToken, curVault);
            // Update the amount of all funds in the token pool
            withdrawed = withdrawed.add(amount);
            _VTS.setIPWithdrawed(_ipToken, _baseToken, withdrawed);
            // Update the amount withdrawn
            return amount;
        }
    }
```

Recommendation： nothing.

3.11. GPDepositRaising logic design【PASS】

Perform security audits on the logic design of the GP in the contract investing funds in the token pool RAISING period to check whether there is an integer overflow or a logic design error.

Audit analysis: The logical design of the deposited funds is reasonable and correct, and no related security risks have been found.

```
function GPDepositRaising(
        address _ipToken,    //  _ipoken address
        address _baseToken, //  _baseToken address
        uint256 _baseTokensAmount,    //  _baseTokensAmount amount
        bool _create      //  whether it is a new user
    )
        external whenNotPaused
        returns (uint256 amount)
    {
        (bool status, bytes memory data) = curVersion.gpdc.delegatecall(
            abi.encodeWithSelector(bytes4(keccak256(
            "GPDepositRaising(address,address,uint256,bool)")),
            _ipToken, _baseToken, _baseTokensAmount, _create));
        require(status == true, "GP Deposit Failed");
        amount = data.bytesToUint256();
        emit RaisingGPDeposit(_ipToken, _baseToken, _baseTokensAmount);
        return amount;
    }
    function GPDepositRaising(
        address _ipToken,
        address _baseToken,
        uint256 _baseTokensAmount,
        bool _create
    )
        external
        lockPool(_ipToken, _baseToken)      //  Lock the token pool
```

```
        returns (uint256 amount)
    {
        poolAtStage(_ipToken, _baseToken, Stages.RAISING);   //  Check the status of the
token pool
        address _gp = msg.sender;
        uint256 oriGPAmount = _GPS.getCurGPAmount(_ipToken, _baseToken);

        amount = _baseTokensAmount;
        IERC20(_baseToken).safeTransferFrom(_gp, address(this), amount);
       //  Transfer _baseToken funds to this contract
        _GPS.setCurGPAmount(_ipToken, _baseToken, oriGPAmount.add(amount));
       //  Update the total number of GPs funds
        _GPS.setCurGPBalance(_ipToken, _baseToken, oriGPAmount.add(amount));
       //  Update the total fund balance of GPs
        if (_create) {
            _GPS.insertGP(_ipToken, _baseToken, _gp, amount, false);
            _GPS.setGPBaseBalance(_ipToken, _baseToken, _gp, amount);
        } else {
            uint256 oriAmount = _GPS.getGPBaseAmount(_ipToken, _baseToken, _gp);
            _GPS.setGPBaseAmount(_ipToken, _baseToken, _gp, oriAmount.add(amount));
            _GPS.setGPBaseBalance(_ipToken, _baseToken, _gp, oriAmount.add(amount));
        }
       //  Update the GP quantity and balance of the user's account
        require(amount > 0, "Deposit Zero");
        return amount;
    }
    function insertGP(address _ipt, address _bst, address _gp, uint256 _amount, bool running)
external onlyProxy {
        require(!pools[_ipt][_bst].GPM[_gp].valid, "GP Already Exist");
        pools[_ipt][_bst].GPA.push(_gp);

        pools[_ipt][_bst].GPM[_gp].valid = true;
        pools[_ipt][_bst].GPM[_gp].id = pools[_ipt][_bst].GPA.length;
```

```
        if (running) {
            pools[_ipt][_bst].GPM[_gp].baseTokensAmount = 0;
            pools[_ipt][_bst].GPM[_gp].runningDepositAmount = _amount;
        } else {
            pools[_ipt][_bst].GPM[_gp].baseTokensAmount = _amount;
            pools[_ipt][_bst].GPM[_gp].runningDepositAmount = 0;
        }

        pools[_ipt][_bst].GPM[_gp].ipTokensAmount = 0;
        pools[_ipt][_bst].GPM[_gp].raisedFromLPAmount = 0;
        pools[_ipt][_bst].GPM[_gp].baseTokensBalance = 0;
    }
    function setGPBaseBalance(address _ipt, address _bst, address _gp, uint256 _amount) external
onlyProxy {
        require(pools[_ipt][_bst].GPM[_gp].valid, "GP Not Exist");
        pools[_ipt][_bst].GPM[_gp].baseTokensBalance = _amount;
    }
    function setGPBaseAmount(address _ipt, address _bst, address _gp, uint256 _amount) external
onlyProxy {
        require(pools[_ipt][_bst].GPM[_gp].valid, "GP Not Exist");
        pools[_ipt][_bst].GPM[_gp].baseTokensAmount = _amount;
    }
```

Recommendation： nothing.

3.12. GPDepositRunning logic design【PASS】

Perform security audits on the logic design of the GPs in the contract investing funds during the token pool RUNNING period to check whether there is an integer overflow or logic design errors.

Audit analysis: The investment logic design is reasonable and correct, and no related security risks are found.

```
    function GPDepositRunning((
```

```
            address _ipToken,    //  _ipoken address
            address _baseToken, //  _baseToken address
            uint256 _baseTokensAmount,   //  _baseTokensAmount amount
            bool _create      //  whether it is a new user
        )
            external whenNotPaused
            returns (uint256 amount)
        {
            (bool status, bytes memory data) = curVersion.gpdc.delegatecall(
                abi.encodeWithSelector(bytes4(keccak256(
                "GPDepositRunning(address,address,uint256,bool)")),
                _ipToken, _baseToken, _baseTokensAmount, _create));
            //  Call the GPDpositRunning function of GPDpositLogic.sol
            require(status == true, "GP Deposit Failed");
            //  Determine whether the call is successful
            amount = data.bytesToUint256();
            emit RunningGPDeposit(_ipToken, _baseToken, _baseTokensAmount);
            //  Log event
            return amount;
        }
    function GPDepositRunning(
            address _ipToken,
            address _baseToken,
            uint256 _baseTokensAmount,
            bool _create
        )
            external
            lockPool(_ipToken, _baseToken)      //  Lock the token pool
            returns (uint256 amount)
        {
            poolAtStage(_ipToken, _baseToken, Stages.RUNNING);        //  Determine the status
of the token pool
            address _gp = msg.sender;
```

```
        uint256 maxAmount = updateMaxIPCanRaise(_ipToken, _baseToken);
      // Query can increase the upper limit
        uint256 curAmount = _GPS.getCurGPAmount(_ipToken, _baseToken);
          // Query the current quantity
        require(maxAmount > curAmount, "No Space Left");
      // Ensure that the upper limit is greater than the current quantity
        amount = maxAmount - curAmount > _baseTokensAmount ?
                    _baseTokensAmount : maxAmount - curAmount;
      // If the amount is greater than the upper limit that can be deposited, the amount is set to
the value that can be deposited
        IERC20(_baseToken).safeTransferFrom(_gp, address(this), amount);
      // Transfer _baseToken funds to this contract
        if (_create) {
            _GPS.insertGP(_ipToken, _baseToken, _gp, amount, true);
        } else {
      uint256 oriAmount = _GPS.getGPRunningDepositAmount(_ipToken, _baseToken, _gp);
      require(oriAmount == 0, "Wait To Do");
      _GPS.setGPRunningDepositAmount(_ipToken, _baseToken, _gp, amount);
        }
      // Update the number of GPs in this account
        require(amount > 0, "Deposit Zero");
        return amount;
    }
    function updateMaxIPCanRaise(
        address _ipToken,
        address _baseToken
    )
        private
        returns (uint256 maxAmount)
    {
        // Max amount of baseToken IP can raise changes according to the token price
        uint256 inUnit = 10**ERC20(_baseToken).decimals();
        uint256 price = NPSwap.getAmountOut(_baseToken, _ipToken, inUnit);
```

```
                uint256 IPStake = _IPS.getIPTokensAmount(_ipToken, _baseToken);
                uint256 initPrice = _IPS.getPoolInitPrice(_ipToken, _baseToken);
                uint32 impawnRatio = _IPS.getIPImpawnRatio(_ipToken, _baseToken);
                // part 1
                uint256 amount =
IPStake.mul(impawnRatio).div(RATIO_FACTOR).mul(price.sqrt()).div(initPrice.sqrt()).mul(inUnit).d
iv(initPrice);
                maxAmount = amount;
                // part2
                amount =
IPStake.mul(impawnRatio).div(RATIO_FACTOR).mul(alpha).div(RATIO_FACTOR).mul(inUnit).di
v(initPrice);
                amount = amount.mul(price).div(initPrice);
                maxAmount = maxAmount.add(amount);

                amount = _IPS.getIPInitCanRaise(_ipToken, _baseToken);
                maxAmount = maxAmount > amount ? maxAmount : amount;
                _IPS.setIPMaxCanRaise(_ipToken, _baseToken, maxAmount);
            }
```

Recommendation： nothing.

3.13. GPDoDepositRunning logic design【PASS】

Perform security audits on the logic design of the GPs in the contract investing funds during the token pool RUNNING period to check whether there is an integer overflow or logic design errors.

Audit analysis: The investment logic design is reasonable and correct, and no related security risks are found.

```
            function GPDoDepositRunning(
                address _ipToken,
                address _baseToken
            }
```

```
        external
        lockPool(_ipToken, _baseToken)
    {
        poolAtStage(_ipToken, _baseToken, Stages.RUNNING);
        address _gp = msg.sender;
        uint256 amount = _GPS.getGPRunningDepositAmount(_ipToken, _baseToken, _gp);

        require(amount > 0, "Already done");
        _GPS.setCurGPAmount(_ipToken, _baseToken,
                                _GPS.getCurGPAmount(_ipToken,
_baseToken).add(amount));
        _GPS.setGPBaseAmount(_ipToken, _baseToken, _gp,
                                _GPS.getGPBaseAmount(_ipToken, _baseToken,
_gp).add(amount));
        _GPS.setGPRunningDepositAmount(_ipToken, _baseToken, _gp, 0);
        // Update balance should before raise from LP.
        updateGPBalance(_ipToken, _baseToken);
        uint256 _amount = amount.sub(chargeVaultFee(_ipToken, _baseToken, amount));
        uint256 raiseLP = raiseFromLP(_ipToken, _baseToken, _amount);
        uint256 oriBalance = _GPS.getGPBaseBalance(_ipToken, _baseToken, _gp);
        _GPS.setGPBaseBalance(_ipToken, _baseToken, _gp, oriBalance.add(_amount));
        oriBalance = _GPS.getCurGPBalance(_ipToken, _baseToken);
        _GPS.setCurGPBalance(_ipToken, _baseToken, oriBalance.add(_amount));

        uint256 swappedIP = NPSwap.swap(_baseToken, _ipToken,
                                        _amount.add(raiseLP));
        oriBalance = _GPS.getCurIPAmount(_ipToken, _baseToken);
        _GPS.setCurIPAmount(_ipToken, _baseToken, oriBalance.add(swappedIP));
        allocateFunds(_ipToken, _baseToken);
    }
function chargeVaultFee(
    address _ipToken,
    address _baseToken,
```

```
        uint256 _amount
    )
        private
        returns (uint256 fee)
    {
        // Part of GP investment would be transfered into vault as fee
        uint32 chargeRatio = _IPS.getIPChargeRatio(_ipToken, _baseToken);

        fee = _amount.mul(chargeRatio).div(RATIO_FACTOR);
        _VTS.setTotalVault(_ipToken, _baseToken,
                           _VTS.getTotalVault(_ipToken, _baseToken).add(fee));
        _VTS.setCurVault(_ipToken, _baseToken,
                         _VTS.getCurVault(_ipToken, _baseToken).add(fee));
        return fee;
    }
    function raiseFromLP(
        address _ipToken,
        address _baseToken,
        uint256 _amount
    )
        private
        returns (uint256 amount)
    {
        uint256 curLPAmount = _LPS.getCurLPAmount(_ipToken, _baseToken);
        uint256 curRaiseLP = _GPS.getCurRaiseLPAmount(_ipToken, _baseToken);
        amount = _amount.mul(raiseRatio).div(RATIO_FACTOR);
        amount = amount > curLPAmount.sub(curRaiseLP) ?
                          curLPAmount.sub(curRaiseLP) : amount;
        _GPS.setCurRaiseLPAmount(_ipToken, _baseToken, curRaiseLP.add(amount));
        return amount;
    }
    function updateGPBalance(
        address _ipToken,
```

```
        address _baseToken
    )
        private
    {
        uint256 IPAmount = _GPS.getCurIPAmount(_ipToken, _baseToken);
        if (IPAmount == 0) {
            // No GP in this pool before, return directly.
            return;
        }
        uint256 inUnit = 10**ERC20(_ipToken).decimals();
        uint256 price = NPSwap.getAmountOut(_ipToken, _baseToken, inUnit);
        uint256 len = _GPS.getGPArrayLength(_ipToken, _baseToken);
        // If sub fail, the pool should do GP liquidation.
        uint256 balance =
IPAmount.mul(price).div(inUnit).sub(_GPS.getCurRaiseLPAmount(_ipToken, _baseToken));
        uint256 resBalance = balance;

        _GPS.setCurGPBalance(_ipToken, _baseToken, balance);
        for (uint256 i = 0; i < len; i++) {
            address gp = _GPS.getGPByIndex(_ipToken, _baseToken, i);
            uint256 amount = _GPS.getGPHoldIPAmount(_ipToken, _baseToken, gp);
            uint256 curBalance = amount.mul(balance).div(IPAmount);
            resBalance -= curBalance;
            curBalance = i == len - 1 ? curBalance.add(resBalance) : curBalance;
            _GPS.setGPBaseBalance(_ipToken, _baseToken, gp, curBalance);
        }
    }
    function allocateFunds(
        address _ipToken,
        address _baseToken
    )
        private
    {
```

```
        uint256 len = _GPS.getGPArrayLength(_ipToken, _baseToken);
        uint256 balance = _GPS.getCurGPBalance(_ipToken, _baseToken);
        uint256 IPAmount = _GPS.getCurIPAmount(_ipToken, _baseToken);
        uint256 raiseLP = _GPS.getCurRaiseLPAmount(_ipToken, _baseToken);
        uint256 resIPAmount = IPAmount;
        uint256 resRaiseLP = raiseLP;

        for (uint256 i = 0; i < len; i++) {
            address gp = _GPS.getGPByIndex(_ipToken, _baseToken, i);
            uint256 gpBalance = _GPS.getGPBaseBalance(_ipToken, _baseToken, gp);

            uint256 curAmount = gpBalance.mul(IPAmount).div(balance);
            resIPAmount -= curAmount;
            curAmount = i == len - 1 ? curAmount.add(resIPAmount) : curAmount;
            _GPS.setGPHoldIPAmount(_ipToken, _baseToken, gp, curAmount);

            curAmount = gpBalance.mul(raiseLP).div(balance);
            resRaiseLP -= curAmount;
            curAmount = i == len - 1 ? curAmount.add(resRaiseLP) : curAmount;
            _GPS.setGPRaiseLPAmount(_ipToken, _baseToken, gp, curAmount);
        }
    }
```

Recommendation： nothing.

3.14. GPWithdrawRunning logic design【PASS】

Perform security audits on the logic design of GPs withdrawal funds in the contract to check whether there are integer overflows, reentry attacks, or logic design errors.

Audit analysis: The logic design of the withdrawal of funds is reasonable and correct, and no related security risks are found.

```
function GPWithdrawRunning(
```

```
        address _ipToken,     //  _ipToken address
        address _baseToken, //  _baseToken address
        uint256 _baseTokensAmount    //  _baseTokens number
    )
        external whenNotPaused
        returns (uint256 amount)
    {
        (bool status, bytes memory data) = curVersion.gpwc.delegatecall(
            abi.encodeWithSelector(bytes4(keccak256(
            "GPWithdrawRunning(address,address,uint256)")),
            _ipToken, _baseToken, _baseTokensAmount));
       //  Call the GPWithdrawRunning function of GPWithdrawLogic.sol
        require(status == true, "GP Withdraw Failed");
        //  Judge whether it is successful
        amount = data.bytesToUint256();
        emit RunningGPWithdraw(_ipToken, _baseToken, _baseTokensAmount);
        //  Record withdrawal events
        return amount;
    }
    function GPWithdrawRunning(
            address _ipToken,
            address _baseToken,
            uint256 _baseTokensAmount
        )
            external
            lockPool(_ipToken, _baseToken)      //  Lock the token pool
            returns (uint256 amount)
        {
            address _gp = msg.sender;
            require(_GPS.getGPValid(_ipToken, _baseToken, _gp) == true, "GP Not Exist");
           //  Determine whether the account is available
            uint256 belongLP = _GPS.getGPRaiseLPAmount(_ipToken, _baseToken, _gp);
           //  Get the number of fixed deposits in the account
```

```
                uint256 IPAmount = _GPS.getGPHoldIPAmount(_ipToken, _baseToken, _gp);
               //  Get the number of account_ipToken
                uint256 swappedBase = NPSwap.swap(_ipToken, _baseToken, IPAmount);
               //  Get the amount of IPAmount _ipToken converted into the amount of _baseToken
                amount = swappedBase > belongLP ? swappedBase.sub(belongLP) : 0;
               //  Determine that the number of swappedBase is greater than the number of account
basetoken
                uint256 GPBase = _GPS.getGPBaseAmount(_ipToken, _baseToken, _gp);
               //  Get the number of GPBase
                uint256 earnedGP = amount > GPBase ? amount.sub(GPBase) : 0;
               //  The amount is required to be greater than GPBase, otherwise the earnedGP will be
reset to zero
                if (earnedGP > 0) {
                    IERC20(_baseToken).safeTransfer(DGTBeneficiary, earnedGP.mul(20).div(100));
                   //  Deduct one-fifth of earnedGP to DGT Beneficiary
                    amount = amount.sub(earnedGP.mul(20).div(100));
                }
                if (amount > 0) {
                    IERC20(_baseToken).safeTransfer(_gp, amount);
               //  Send the rest to the withdrawal
                }
                uint256 gpAmount = _GPS.getGPBaseAmount(_ipToken, _baseToken, _gp);
                uint256 poolAmount = _GPS.getCurGPAmount(_ipToken, _baseToken);
                _GPS.setCurGPAmount(_ipToken, _baseToken, poolAmount.sub(gpAmount));
               //  Update GPAmoun
                gpAmount = _GPS.getGPBaseBalance(_ipToken, _baseToken, _gp);
                poolAmount = _GPS.getCurGPBalance(_ipToken, _baseToken);
                _GPS.setCurGPBalance(_ipToken, _baseToken, poolAmount.sub(gpAmount));
               //  Update GPBalance
                gpAmount = _GPS.getGPHoldIPAmount(_ipToken, _baseToken, _gp);
                poolAmount = _GPS.getCurIPAmount(_ipToken, _baseToken);
                _GPS.setCurIPAmount(_ipToken, _baseToken, poolAmount.sub(gpAmount));
                  //  Update IPAmount
```

```
            gpAmount = _GPS.getGPRaiseLPAmount(_ipToken, _baseToken, _gp);
            poolAmount = _GPS.getCurRaiseLPAmount(_ipToken, _baseToken);
            _GPS.setCurRaiseLPAmount(_ipToken, _baseToken, poolAmount.sub(gpAmount));
            // Update CurRaiseLPAmount
            _GPS.deleteGP(_ipToken, _baseToken, _gp);
            // delete users
            return amount;
        }
    }
    function deleteGP(address _ipt, address _bst, address _gp) external onlyProxy {
        require(pools[_ipt][_bst].GPM[_gp].valid, "GP Not Exist");
        uint256 id = pools[_ipt][_bst].GPM[_gp].id;
        uint256 length = pools[_ipt][_bst].GPA.length;

        pools[_ipt][_bst].GPA[id - 1] = pools[_ipt][_bst].GPA[length - 1];
        pools[_ipt][_bst].GPM[pools[_ipt][_bst].GPA[length - 1]].id = id;
        pools[_ipt][_bst].GPA.pop();
        pools[_ipt][_bst].GPM[_gp].valid = false;
        pools[_ipt][_bst].GPM[_gp].id = 0;
        pools[_ipt][_bst].GPM[_gp].baseTokensAmount = 0;
        pools[_ipt][_bst].GPM[_gp].runningDepositAmount = 0;
        pools[_ipt][_bst].GPM[_gp].ipTokensAmount = 0;
        pools[_ipt][_bst].GPM[_gp].raisedFromLPAmount = 0;
        pools[_ipt][_bst].GPM[_gp].baseTokensBalance = 0;
    }
```

Recommendation： nothing.

3.15. computeVaultReward logic design【PASS】

Perform security audits on the design of the computeVaultReward reward calculation logic in the contract to check whether there are integer overflows, reentry attacks, or logic design errors.

Audit analysis: The reward calculation logic design is reasonable and correct, and no related security risks are found.

```
function computeVaultReward(
        address _ipToken,
        address _baseToken
)
        external
        lockPool(_ipToken, _baseToken)
{
        poolAtStage(_ipToken, _baseToken, Stages.RUNNING);
        uint256 vault = getVaultReward(_ipToken, _baseToken);
        uint256 len = _LPS.getLPArrayLength(_ipToken, _baseToken);
        uint256 LPAmount = _LPS.getCurLPAmount(_ipToken, _baseToken);
        uint256 resVault = vault;

        for (uint256 i = 0; i < len; i++) {
                address lp = _LPS.getLPByIndex(_ipToken, _baseToken, i);
                uint256 reward = _LPS.getLPVaultReward(_ipToken, _baseToken, lp);
                uint256 amount = _LPS.getLPBaseAmount(_ipToken, _baseToken, lp);
                //reuse local varible
                uint256 curVault = vault.mul(amount).div(LPAmount);
                resVault -= curVault;
                curVault = i == len - 1 ? curVault.add(resVault) : curVault;
                reward = reward.add(curVault);
                _LPS.setLPVaultReward(_ipToken, _baseToken, lp, reward);
        }
}
function getVaultReward(
        address _ipToken,
        address _baseToken
)
        internal
        returns (uint256 vault)
```

```
{
    uint256 lastTime = _VTS.getLastUpdateTime(_ipToken, _baseToken);
    uint256 endTime = _IPS.getPoolAuctionEndTime(_ipToken, _baseToken).add(
                        _IPS.getIPDuration(_ipToken, _baseToken));
    uint256 curVault = _VTS.getCurVault(_ipToken, _baseToken);

    require(block.timestamp > lastTime && block.timestamp < endTime,
            "Timestamp Incorrect");
    vault = curVault.mul(block.timestamp.sub(lastTime)).div(endTime.sub(lastTime));
    _VTS.setCurVault(_ipToken, _baseToken, curVault.sub(vault));
    _VTS.setLastUpdateTime(_ipToken, _baseToken, block.timestamp);
}
```

Recommendation： nothing.

4. Basic code vulnerability detection

4.1. Compiler version security【PASS】

Check whether a safe compiler version is used in the contract code implementation.

Audit result: After testing, the compiler version 0.8.0 is formulated in the smart contract code, and there is no such security issue.

Recommendation： nothing.

4.2. Redundant code【PASS】

Check whether the contract code implementation contains redundant code.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.3. Use of safe arithmetic library【PASS】

Check whether the SafeMath safe arithmetic library is used in the contract code implementation.

Audit result: After testing, the SafeMath safe arithmetic library has been used in the smart contract code, and there is no such security problem.

Recommendation： nothing.

4.4. Not recommended encoding【PASS】

Check whether there is an encoding method that is not officially recommended or abandoned in the contract code implementation

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.5. Reasonable use of require/assert【PASS】

Check the rationality of the use of require and assert statements in the contract code implementation.

Audit result: After testing, the require statement is reasonably used and the assert statement is not used in the smart contract code, and there is no such security problem.

Recommendation： nothing.

4.6. fallback function safety【PASS】

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

Audit result: After testing, the fallback function is not defined in the smart contract code, and there is no such security problem.

Recommendation： nothing.

4.7. 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 a smart contract makes the contract vulnerable to attacks like phishing.

Audit result: After testing, the tx.origin global variable is not used in the smart contract code, and there is no such security problem.

Recommendation： nothing.

4.8. Owner permission control【PASS】

Check whether the owner in the contract code implementation has excessive authority. For example, arbitrarily modify other account balances, etc.

Audit result: After testing, the security owner library is used in the smart contract code, and there is no such security problem.

Recommendation： nothing.

4.9. Gas consumption detection【PASS】

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

Audit result: After testing, there is no circular call in the smart contract code, and there is no such security problem.

Recommendation： nothing.

4.10. call injection attack【PASS】

When the call function is called, strict permission control should be done, or the function called by the call should be written dead.

Audit result: After testing, the smart contract does not use the call function, and this vulnerability does not exist.

Recommendation： nothing.

4.11. Low-level function safety【PASS】

Check whether there are security vulnerabilities in the use of low-level functions (call/delegatecall) in the contract code implementation

The execution context of the call function is in the called contract; the execution context of the delegatecall function is in the contract that currently calls the function.

Audit result: After testing, the delegatecall function in the smart contract code performs the operation correctly, and there is no such security problem.

Recommendation： nothing.

4.12. Vulnerability of additional token issuance【PASS】

Check whether there is a function that may increase the total amount of tokens in the token contract after initializing the total amount of tokens.

Audit result: After testing, the smart contract code does not have the function of issuing additional tokens, so it is approved.

Recommendation： nothing.

4.13. Access control defect detection【PASS】

Different functions in the contract should set reasonable permissions.

Check whether each function in the contract correctly uses keywords such as public and private for visibility modification, check whether the contract is correctly defined and use modifier to restrict access to key functions to avoid problems caused by unauthorized access.

Audit result: After testing, keywords are reasonably used in the smart contract code to modify the function, and there is no such security problem.

Recommendation： nothing.

4.14. Numerical overflow detection【PASS】

The arithmetic problems in smart contracts refer to integer overflow and integer underflow.

Solidity can handle up to 256-bit numbers (2^256-1). If the maximum number increases by 1, it will overflow to 0. Similarly, when the number is an unsigned type, 0 minus 1 will underflow to get the maximum digital value.

Integer overflow and underflow are not a new type of vulnerability, but they are especially dangerous in smart contracts. Overflow conditions can lead to incorrect results, especially if the possibility is not expected, which may affect the reliability and safety of the program.

Audit result: After testing, the meta-calculation operations in the smart contract code all use SafeMath library functions, and there is no such security problem.

Recommendation： nothing.

4.15. Arithmetic accuracy error【PASS】

As a programming language, Solidity has data structure design similar to ordinary programming languages, such as variables, constants, functions, arrays, functions, structures, etc. There is also a big difference between Solidity and ordinary programming languages-Solidity does not float Point type, and all the numerical calculation results of Solidity will only be integers, there will be no decimals, and it is not allowed to define decimal type data. Numerical calculations in the contract are indispensable, and the design of numerical calculations may cause relative errors. For example, the same level of calculations: 5/2*10=20, and 5*10/2=25, resulting in errors, which are larger in data The error will be larger and more obvious.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.16. Incorrect use of random numbers【PASS】

Smart contracts may need to use random numbers. Although the functions and variables provided by Solidity can access values that are obviously unpredictable, such as block.number and block.timestamp, they are usually more public than they

appear or are affected by miners. These random numbers are predictable to a certain extent, so malicious users can usually copy it and rely on its unpredictability to attack the function.

Audit result: After testing, no random number is used in the smart contract code, and there is no such security problem.

Recommendation： nothing.

4.17. Unsafe interface usage【PASS】

Check whether unsafe interfaces are used in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.18. Variable coverage【PASS】

Check whether there are security issues caused by variable coverage in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.19. Uninitialized storage pointer【PASS】

In solidity, a special data structure is allowed to be a struct structure, and the local variables in the function are stored in storage or memory by default.

The existence of storage (memory) and memory (memory) are two different concepts. Solidity allows pointers to point to an uninitialized reference, while uninitialized local storage will cause variables to point to other storage variables, leading to variable coverage, or even more serious As a consequence, you should avoid initializing struct variables in functions during development.

Audit result: After testing, the smart contract code structure is used correctly and there is no such problem.

Recommendation： nothing.

4.20. Return value call verification【PASS】

This problem mostly occurs in smart contracts related to currency transfer, so it is also called silent failed delivery or unchecked delivery.

In Solidity, there are transfer(), send(), call.value() and other currency transfer methods, which can all be used to send Ether to an address. The difference is: When the transfer fails, it will be thrown and the state will be rolled back; Only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when send fails; only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when call.value fails to be sent; all available gas will be passed for calling

(can be Limit by passing in gas_value parameters), which cannot effectively prevent reentry attacks.

If the return value of the above send and call.value transfer functions is not checked in the code, the contract will continue to execute the following code, which may lead to unexpected results due to Ether sending failure.

Audit result: After testing, the smart contract code does not use transfer (), send (), call.value () and other currency transfer methods, and there is no such security problem.

Recommendation： nothing.

4.21. Transaction order dependency【PASS】

Since miners always get gas fees through codes that represent externally owned addresses (EOA), users can specify higher fees for faster transactions. Since the Ethereum blockchain is public, everyone can see the content 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 transaction at a higher fee to preempt the original solution.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.22. Timestamp dependency attack【PASS】

The timestamp of the data block usually uses the local time of the miner, and this time can fluctuate in the range of about 900 seconds. When other nodes accept a new block, it only needs to verify whether the timestamp is later than the previous block and The error with local time is within 900 seconds. A miner can profit from it by setting the timestamp of the block to satisfy the conditions that are beneficial to him as much as possible.

Check whether there are key functions that depend on the timestamp in the contract code implementation.

Audit result: After testing, the correct use of timestamps in the smart contract code does not have this security problem.

Recommendation： nothing.

4.23. Denial of service attack【PASS】

In the world of Ethereum, denial of service is fatal, and a smart contract that has suffered this type of attack may never be able to return to its normal working state. There may be many reasons for the denial of service of the smart contract, including malicious behavior as the transaction recipient, artificially increasing the gas required for computing functions to cause gas exhaustion, abusing access control to access the private component of the smart contract, using confusion and negligence, etc. Wait.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.24. Fake recharge vulnerability【PASS】

The transfer function of the token contract uses the if judgment method to check the balance of the transfer initiator (msg.sender). When balances[msg.sender] <value, enter the else logic part and return false, and finally no exception is thrown. We believe that only if/else this kind of gentle judgment method is an imprecise coding method in sensitive function scenarios such as transfer.

Audit result: After testing, the key judgments in the smart contract code are very strict, and there is no such security problem.

Recommendation： nothing.

4.25. Reentry attack detection【PASS】

The call.value() function in Solidity consumes all the gas it receives when it is used to send Ether. When the call.value() function to send Ether occurs before the actual reduction of the sender's account balance, There is a risk of reentry attacks.

Audit results： After testing, the call function is not used in the smart contract code, and there is no such security problem.

Recommendation： nothing.

4.26. Replay attack detection【PASS】

If the contract involves the need for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks

In the asset management system, there are often cases of entrusted management. The principal assigns assets to the trustee for management, and the principal pays a certain fee to the trustee. This business scenario is also common in smart contracts.

Audit results： After testing, the smart contract does not have this vulnerability.

Recommendation： nothing.

4.27. Rearrangement attack detection【PASS】

A rearrangement attack refers to a miner or other party trying to "compete" with smart contract participants by inserting their own information into a list or mapping (mapping), so that the attacker has the opportunity to store their own information in the contract in.

Audit results： After testing, there are no related vulnerabilities in the smart contract code.

Recommendation： nothing.

5. Appendix A：Vulnerability rating standard

Smart contract vulnerability rating standards	
Level	**Level Description**
High	Vulnerabilities that can directly cause the loss of token contracts or user funds, such as: value overflow loopholes that can cause the value of tokens to zero, fake recharge loopholes that can cause exchanges to lose tokens, and can cause contract accounts to lose ETH or tokens. Access loopholes, etc.; Vulnerabilities that can cause loss of ownership of token contracts, such as: access control defects of key functions, call injection leading to bypassing of access control of key functions, etc.; Vulnerabilities that can cause the token contract to not work properly, such as: denial of service vulnerability caused by sending ETH to malicious addresses, and denial of service vulnerability caused by exhaustion of gas.
Medium	High-risk vulnerabilities that require specific addresses to trigger, such as value overflow vulnerabilities that can be triggered by token contract owners; access control defects for non-critical functions, and logical design defects that cannot cause direct capital losses, etc.
Low	Vulnerabilities that are difficult to be triggered, vulnerabilities with limited damage after triggering, such as value overflow vulnerabilities that require a large amount of ETH or tokens to trigger, vulnerabilities where attackers cannot directly profit after triggering value overflow, and the transaction sequence triggered by specifying high gas depends on the risk.

6. Appendix B：Introduction to auditing tools

6.1. Manticore

Manticore is a symbolic execution tool for analyzing binary files and smart contracts. Manticore includes a symbolic Ethereum Virtual Machine (EVM), an EVM disassembler/assembler and a convenient interface for automatic compilation and analysis of Solidity. It also integrates Ethersplay, Bit of Traits of Bits visual disassembler for EVM bytecode, used for visual analysis. Like binary files, Manticore provides a simple command line interface and a Python for analyzing EVM bytecode API.

6.2. Oyente

Oyente is a smart contract analysis tool. Oyente can be used to detect common bugs in smart contracts, such as reentrancy, transaction sequencing dependencies, etc. More convenient, Oyente's design is modular, so this allows advanced users to implement and Insert their own detection logic to check the custom attributes in their contract.

6.3. securify.sh

Securify can verify common security issues of Ethereum smart contracts, such as disordered transactions and lack of input verification. It analyzes all possible execution paths of the program while fully automated. In addition, Securify also has a

specific language for specifying vulnerabilities, which makes Securify can keep an eye on current security and other reliability issues at any time.

6.4. Echidna

Echidna is a Haskell library designed for fuzzing EVM code.

6.5. MAIAN

MAIAN is an automated tool for finding vulnerabilities in Ethereum smart contracts. Maian processes the bytecode of the contract and tries to establish a series of transactions to find and confirm the error.

6.6. ethersplay

ethersplay is an EVM disassembler, which contains relevant analysis tools.

6.7. ida-evm

ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

6.8. Remix-ide

ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

6.9. Knownsec Penetration Tester Special Toolkit

Pen-Tester tools collection is created by KnownSec team. It contains plenty of Pen-Testing tools such as automatic testing tool, scripting tool, Self-developed tools etc.

KNOWNSEC

Beijing KnownSec Information Technology Co., Ltd.

Advisory telephone	+86(10)400 060 9587
E-mail	sec@knownsec.com
Website	www.knownsec.com
Address	wangjing soho T2-B2509,Chaoyang District, Beijing