Teeter (TeeterUnderlyingTop) Contract

Smart Contract Audit

- Teeter Audit Summary
- Teeter Audit
  - Document information
    - Audit results
    - Audited target file
  - Vulnerability analysis
    - Vulnerability distribution
    - Summary of audit results
    - Contract file
    - Analysis of audit results
      - Re-Entrancy
      - Arithmetic Over/Under Flows
      - Unexpected Blockchain Currency
      - Delegatecall
      - Default Visibilities
      - Entropy Illusion
      - External Contract Referencing
      - Unsolved TODO comments
      - Short Address/Parameter Attack
      - Unchecked CALL Return Values
      - Race Conditions / Front Running
      - Denial Of Service (DOS)
      - Block Timestamp Manipulation
      - Constructors with Care
      - Unintialised Storage Pointers
      - Floating Points and Numerical Precision
      - tx.origin Authentication
      - Permission restrictions

Teeter Audit Summary

Project name : Teeter (TeeterUnderlyingTop) Contract

Project address: None

Code URL : https://github.com/teeter-finance/teeter-contracts/blob/main/TeeterUnderlyingTop.sol

Commit : d22187f3b5a15845f343ff402aed97aa3d43ab2c

Project target : Teeter Contract Audit

Blockchain : Binance Smart Chain（BSC）

Test result : PASSED

Audit Info

Audit NO : 0X202201110016

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

Teeter Audit

The Teeter team asked us to review and audit their Teeter contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
Teeter Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2022-01-11

Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Teeter contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 3 months from the date of output.

Disclaimer

Armors Labs Reports is not and should not be regarded as an "approval" or "disapproval" of any particular project or team. These reports are not and should not be regarded as indicators of the economy or value of any "product" or "asset" created by any team. Armors do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'…)

Armors Labs Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Armors does not guarantee the safety or functionality of the technology agreed to be analyzed.

Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused. Armors Labs Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

Audited target file

file	md5
Teeter.sol	bdc2c8b181051f6ac00e8c2077b4089a

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe

Vulnerability	status
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe
Permission restrictions	safe

Contract file

Teeter.sol

```
pragma solidity =0.5.16;

import "./interfaces/ITeeterUnderlyingTop.sol";
import "./interfaces/ITeeterFactory.sol";
import "./TeeterERC20.sol";
import "./libraries/SafeMath.sol";
import "./libraries/TeeterLibrary.sol";
import "./interfaces/IERC20.sol";
import './interfaces/ITeeterLeverage.sol';
import './libraries/TransferHelper.sol';

contract TeeterUnderlyingTop is ITeeterUnderlyingTop, TeeterERC20 {
    address[] public exchangeAddrs;
    address public factory;
    address public leverage;
    address public token0;
    uint256 public capAddU;
    uint256 public underlyingU;
    uint256 private underlyingQTY;
    uint256 private underlyingValueEn;
    uint256 private fundSoldQTY;
    uint256 private fundSoldValueEn;
    uint256 public price0En;
    uint256 public priceEn;
    uint256 public purcRateEn;
    uint256 public redeeRateEn;
    uint256 public manaRateEn;
    uint256 public liquDiscountRateEn;
    uint256 public ownerRateEn;
    address public addrBase;
    uint256 public balBaseLast;
    uint256 private nvEn = 51922968585348276285304963292200960;
    uint8 public initLever;
    uint256 private presLeverEn;
    uint8 public direction;

    uint8 public status = 1;
    uint private blockTimestampLast;
    uint public blockTimestampInit;
    uint256 private capPoolU;
    uint256 private feeU;
    uint256 private usrMarginU;
    uint256 private capU;
    uint256 public refNvEn = 51922968585348276285304963292200960;
    uint256 private usrPoolQ;
```

```
    uint256 public ownerU;

    uint256 public exePrice;

    uint256 private unlocked = 1;

    constructor() public {
        factory = msg.sender;
    }

    modifier lock() {
        require(unlocked == 1, "TeeterUnderlyingTOP: LOCKED");
        unlocked = 0;
        _;
        unlocked = 1;
    }


    function initialize(
        address _token0, uint8 _lever, uint8 _direction,
        address _leverage, uint256 _purcRateEn, uint256 _redeeRateEn, uint256 _manaRateEn,
        address _addrBase, uint256 _liquDiscountRateEn, uint256 _ownerRateEn, address _pair
        ) external {
        require(msg.sender == factory, "TeeterUnderlyingTOP: FORBIDDEN");
        token0 = _token0;
        initLever = _lever;
        presLeverEn = uint256(_lever)<<112;
        direction = _direction;
        leverage = _leverage;
        purcRateEn = _purcRateEn;
        redeeRateEn = _redeeRateEn;
        manaRateEn = _manaRateEn;
        addrBase = _addrBase;
        liquDiscountRateEn = _liquDiscountRateEn;
        ownerRateEn = _ownerRateEn;
        symbol = 'bTeeter';
        exchangeAddrs.push(_pair);
        exchangeAddrs.push(address(0x0));
    }

    function updateParameter(
        uint256 _purcRateEn, uint256 _redeeRateEn, uint256 _manaRateEn, uint256 _liquDiscountRateEn,
        uint256 _ownerRateEn, address exchangeAddr0, address exchangeAddr1
        ) external {
        require(msg.sender == ITeeterFactory(factory).owner(), "TeeterUnderlyingTOP: FORBIDDEN");
        purcRateEn = _purcRateEn;
        redeeRateEn = _redeeRateEn;
        manaRateEn = _manaRateEn;
        liquDiscountRateEn = _liquDiscountRateEn;
        ownerRateEn = _ownerRateEn;
        exchangeAddrs[0] = exchangeAddr0;
        if(exchangeAddr1!=address(0x0)){exchangeAddrs[1] = exchangeAddr1;}
    }


    function getReserves() public view returns (
        uint256 _fundSoldValueEn,
        uint256 _fundSoldQTY,
        uint256 _nvEn,
        uint256 _presLeverEn,
        uint256 _underlyingU,
        uint256 _underlyingQTY,
        uint256 _priceEn,
        uint256 _underlyingValueEn,
        uint256 _capPoolU,
        uint256 _usrMarginU,
```

```
        uint256 _feeU,
        uint256 _capU,
        uint256 _usrPoolQ){
        _fundSoldValueEn = fundSoldValueEn;
        _fundSoldQTY = fundSoldQTY;
        _nvEn  = nvEn;
        _presLeverEn = presLeverEn;
        _underlyingU = underlyingU;
        _underlyingQTY = underlyingQTY;
        _priceEn = priceEn;
        _underlyingValueEn = underlyingValueEn;
        _capPoolU = capPoolU;
        _usrMarginU = usrMarginU;
        _feeU = feeU;
        _capU = capU;
        _usrPoolQ = usrPoolQ;
    }

    function updatePrice()public {
        priceEn = TeeterLibrary.getLastPriceEn(token0, exchangeAddrs);
    }

    function _rebalanceD()private{
        presLeverEn = uint256(initLever)<<112;
        refNvEn = nvEn;
        usrMarginU = SafeMath.sub(usrMarginU, capPoolU);
        uint256 underlyingQTYlast = underlyingQTY;
        underlyingQTY = SafeMath.div(SafeMath.mul(usrMarginU, presLeverEn), priceEn);
        if(underlyingQTY > underlyingQTYlast){return;}
        underlyingValueEn = SafeMath.mul(underlyingQTY, priceEn);
        _swap((underlyingQTYlast-underlyingQTY), token0);
        uint256 swapU = SafeMath.sub(
            TeeterLibrary.convertTo18(addrBase, IERC20(addrBase).balanceOf(address(this))), balBaseLa
        );
        balBaseLast += swapU;
        underlyingU += swapU;
        capU += capPoolU;
        capPoolU = 0;

    }

    function _rebalanceU() private{
        refNvEn = nvEn;
        uint256 swapQ;
        uint256 sub1En = SafeMath.mul(
            (SafeMath.sub(usrMarginU, capPoolU) + (SafeMath.mul(usrPoolQ, priceEn)>>112)),
            uint256(initLever)<<112
        );
        uint256 underlyingUDelta = SafeMath.sub(sub1En, underlyingValueEn);
        if(underlyingU > underlyingUDelta){
            underlyingU -= underlyingUDelta;
        }else{
            underlyingUDelta = underlyingU;
            underlyingU = 0;
        }
        _swap(underlyingUDelta, addrBase);
        swapQ = SafeMath.sub(
            TeeterLibrary.convertTo18(token0, IERC20(token0).balanceOf(address(this))), underlyingQTY
        );
        balBaseLast = SafeMath.sub(balBaseLast, underlyingUDelta);
        underlyingQTY += swapQ;
        underlyingValueEn = SafeMath.mul(underlyingQTY, priceEn);
        presLeverEn = SafeMath.div(underlyingValueEn, fundSoldValueEn>>112);
        capU += capPoolU;
        capPoolU = 0;
    }
```

```
function _updateIndexes() public{//local, kovan be private
    require(fundSoldQTY >0 || underlyingU>0,"TeeterUnderlyingTOP: EMPTY");
    if(price0En==priceEn || fundSoldValueEn==0){return;}
    if(status == 1){
        uint timeElapsed = SafeMath.sub(block.timestamp, blockTimestampLast);
        uint daysEn = SafeMath.div((timeElapsed)<<112, 86400);
        uint256 feeLast = feeU;
        feeU += (
            SafeMath.mul(
                usrMarginU,
                SafeMath.mul(manaRateEn, daysEn)>>112
            )>>112
        );
        uint256 _underlyingValueLastEn = underlyingValueEn;
        underlyingValueEn = SafeMath.mul(underlyingQTY, priceEn);
        if(priceEn > price0En){
            fundSoldValueEn += SafeMath.sub(underlyingValueEn, _underlyingValueLastEn) ;
        }else{
            fundSoldValueEn = SafeMath.sub(fundSoldValueEn, SafeMath.sub(_underlyingValueLastEn,
        }
        fundSoldValueEn = SafeMath.sub(fundSoldValueEn, (SafeMath.sub(feeU, feeLast))<<112);
        if(fundSoldValueEn>underlyingValueEn){
            status = 0;
            ownerU += (SafeMath.mul(
                feeU,
                SafeMath.sub(519229685853482762853049632922009 6, ownerRateEn))>>112
            );
            capU = SafeMath.sub(balBaseLast, (ownerU + underlyingU));
            feeU = 0;
            fundSoldValueEn = 0;
            fundSoldQTY = 0;
            nvEn = 0;
            presLeverEn = 0;
            usrMarginU = 0;
            usrPoolQ = 0;
            capPoolU = 0;
            price0En = priceEn;
        }else{
            if(fundSoldQTY!=0){
                nvEn = SafeMath.div(fundSoldValueEn, fundSoldQTY);
                presLeverEn = SafeMath.div(underlyingValueEn, fundSoldValueEn>>112);
            }
            usrMarginU =SafeMath.sub(usrMarginU, (feeU-feeLast));//feeU been added in line 214, s
            uint256 fundSoldValue = fundSoldValueEn>>112;
            if(fundSoldValue >= usrMarginU){
                capPoolU = 0;
            }else{
                capPoolU = usrMarginU - fundSoldValue;
            }
            if(fundSoldValue >= usrMarginU){
                usrPoolQ = SafeMath.div( ((fundSoldValue-usrMarginU)<<112), priceEn);
            }else{
                usrPoolQ = 0;
            }
            price0En = priceEn;
            if(SafeMath.mul(5, nvEn) <= refNvEn){ _rebalanceD(); }else if(SafeMath.div(nvEn, 3) >
        }
    }else{
        price0En = priceEn;
        underlyingValueEn = SafeMath.mul(underlyingQTY, priceEn);
    }
    blockTimestampLast = block.timestamp;
}
```

```
    function mint(address to) external lock returns(uint256 liquidity){
        require(status == 1, "TeeterUnderlyingTOP: FUNDCLOSE");
        uint256 baseBal = TeeterLibrary.convertTo18(addrBase, IERC20(addrBase).balanceOf(address(this
        uint256 baseAmt = SafeMath.sub(baseBal, balBaseLast);
        capAddU += baseAmt;
        require((baseAmt > 10000000), "TeeterUnderlyingTOP: INSUFFICIENT_baseBal");
        updatePrice();
        if(totalSupply == 0){
            require(priceEn!=0, "TEETER_PRICEIS0");
            nvEn = 5192296858534827628530496329220096;
            presLeverEn =1557689057560448288559148898766028 8;
            liquidity = baseAmt;
            underlyingU = baseAmt;
            blockTimestampLast = block.timestamp;
            blockTimestampInit = block.timestamp;
            price0En = priceEn;
        }else{
            _updateIndexes();
            if(status != 1){TransferHelper.safeTransfer(addrBase, to, TeeterLibrary.convert18ToOri(ad
            uint256 feeUCaked = SafeMath.mul(feeU, SafeMath.sub(5192296858534827628530496329220096, o
            uint256 numerator = SafeMath.mul(baseAmt, totalSupply);
            uint256 sub1 = (underlyingValueEn>>112) + capPoolU + feeUCaked + capU + underlyingU;
            uint256 sub2 = SafeMath.mul(usrPoolQ, priceEn)>>112;
            uint256 denominator = SafeMath.sub(sub1, sub2);
            liquidity = SafeMath.div(numerator, denominator);
            require(liquidity !=0, 'TeeterUnderlying:LPTERR');
            underlyingU += baseAmt;
        }
        balBaseLast = baseBal;
        _mint(to, liquidity);
    }


    function purchase(address to) external lock returns(uint256 amtLever){
        require(status == 1, "TeeterUnderlyingTOP: FUNDCLOSE");
        updatePrice();
        uint256 balanceBase = TeeterLibrary.convertTo18(addrBase, IERC20(addrBase).balanceOf(address(
        require(balanceBase != 0, "TeeterUnderlyingTOP: balanceBaseIS0");
        uint256 amtTokenIn = SafeMath.sub(balanceBase, balBaseLast);
        require((amtTokenIn > 1000000000000000000), "TeeterUnderlyingTOP: INSUFFICIENT_AMTTOKENIN");
        _updateIndexes();
        if(status != 1){TransferHelper.safeTransfer(addrBase, to, TeeterLibrary.convert18ToOri(addrBa
        uint256 amtMaxPurc = SafeMath.div(underlyingU<<112, presLeverEn);
        if(amtTokenIn > amtMaxPurc){
            uint256 amtTokenInReturn = TeeterLibrary.convert18ToOri(addrBase, SafeMath.sub(amtTokenIn
            TransferHelper.safeTransfer(addrBase, to, amtTokenInReturn);
            balanceBase -= (amtTokenIn - amtMaxPurc);
            amtTokenIn = amtMaxPurc;
        }
        uint256 feeUDelta = SafeMath.mul(amtTokenIn, purcRateEn)>>112;
        feeU += feeUDelta;
        uint256 amtTokenInFeed = SafeMath.sub(amtTokenIn, feeUDelta);
        uint256 deltaUnderlyingU = SafeMath.mul(amtTokenInFeed, presLeverEn)>>112;
        underlyingU = SafeMath.sub(underlyingU, deltaUnderlyingU) ;
        _swap(deltaUnderlyingU, addrBase);
        uint256 deltaUnderlyingQTY = SafeMath.sub(
            TeeterLibrary.convertTo18(token0, IERC20(token0).balanceOf(address(this))), underlyingQTY
        );
        underlyingQTY += deltaUnderlyingQTY;
        underlyingValueEn = SafeMath.mul(underlyingQTY, priceEn);
        uint256 amtTokenInFeedLeveEn = deltaUnderlyingU<<112;
        uint256 deltaUnderlyingQTYValueEn = SafeMath.mul(deltaUnderlyingQTY, priceEn);
        if(amtTokenInFeedLeveEn > deltaUnderlyingQTYValueEn){
            usrMarginU += SafeMath.sub(
                amtTokenInFeed,
                SafeMath.sub(
```

```
                    amtTokenInFeedLeveEn, deltaUnderlyingQTYValueEn
                )>>112
            );
        }else{
            usrMarginU += amtTokenInFeed;
        }
        if(amtTokenInFeedLeveEn > deltaUnderlyingQTYValueEn){
            capU += ((amtTokenInFeedLeveEn - deltaUnderlyingQTYValueEn)>>112);//already judged before
        }
        amtLever = SafeMath.div(
            SafeMath.sub(
                amtTokenInFeed<<112,
                SafeMath.sub(amtTokenInFeedLeveEn, deltaUnderlyingQTYValueEn)
            ),
            nvEn
        );
        fundSoldQTY += amtLever;
        fundSoldValueEn = SafeMath.mul(fundSoldQTY, nvEn);
        balBaseLast = SafeMath.sub(SafeMath.add(balBaseLast, amtTokenIn), deltaUnderlyingU);//the pri
        ITeeterLeverage(leverage).mint(to, amtLever);
    }

    function _swap(uint256 _amt, address _path0) private {
        address[] memory path = new address[](2);
        if(_path0 == addrBase){
            path[0] = addrBase;
            path[1] = token0;
        }else{
            path[1] = addrBase;
            path[0] = token0;
        }
        uint256 amt = TeeterLibrary.convert18ToOri(path[0], _amt);
        uint256[] memory amounts0;
        uint256[] memory amounts1;
        if(exchangeAddrs[1]!=address(0x0)){
            amounts0 = UniswapV2Library.getAmountsOut(amt/2, path, exchangeAddrs[0]);
            amounts1 = UniswapV2Library.getAmountsOut(SafeMath.sub(amt, amounts0[0]), path, exchangeA
        }else{
            amounts0 = UniswapV2Library.getAmountsOut(amt, path, exchangeAddrs[0]);
            amounts1 = new uint256[](2);
        }
        if(amounts0[0]>0){
            TransferHelper.safeTransfer(path[0], exchangeAddrs[0], amounts0[0]);
            TeeterLibrary.swap(amounts0, path, address(this), exchangeAddrs[0]);
        }
        if(amounts1[0]>0){
            TransferHelper.safeTransfer(path[0], exchangeAddrs[1], amounts1[0]);
            TeeterLibrary.swap(amounts1, path, address(this), exchangeAddrs[1]);
        }

        if(_path0 == addrBase){
            exePrice = SafeMath.div(TeeterLibrary.convertTo18(path[0], amounts0[0]), TeeterLibrary.co
        }else{
            exePrice = SafeMath.div(TeeterLibrary.convertTo18(path[1], amounts0[1]), TeeterLibrary.co
        }
    }

    function redeem(address to) external lock returns(uint256 amtAsset, uint256 amtU){
        require(status == 1, "TeeterUnderlyingTOP: FUNDCLOSE");
        updatePrice();
        uint256 amtLeverTokenIn = IERC20(leverage).balanceOf(address(this));
        require(amtLeverTokenIn !=0, "TeeterUnderlyingTOP: INSUFFICIENT_amtLeverTokenIn");
        _updateIndexes();
        if(status != 1){TransferHelper.safeTransfer(leverage, to, amtLeverTokenIn); return (0, 0);}//
        uint256 feeUDelta =
            SafeMath.mul(
```

```
                SafeMath.mul(amtLeverTokenIn, nvEn)>>112,
                redeeRateEn
            )>>112;
        feeU += feeUDelta;
        uint256 leverTotalSupply = IERC20(leverage).totalSupply();
        uint256 swapU;
        uint256 amtLeverTokenInValue = SafeMath.mul(amtLeverTokenIn, priceEn)>>112;
        underlyingU = underlyingU + SafeMath.div(SafeMath.mul(amtLeverTokenInValue, SafeMath.sub(unde
        uint256 underlyingQTYlast = underlyingQTY;
        underlyingQTY = SafeMath.sub(
            underlyingQTY,
            SafeMath.div(
                SafeMath.mul(underlyingQTY, amtLeverTokenIn), leverTotalSupply
            )
        );

        underlyingValueEn = SafeMath.mul(underlyingQTY, priceEn);
        _swap((underlyingQTYlast-underlyingQTY), token0);// underlyingQTY has been subed, so not need
        swapU = SafeMath.sub(
            TeeterLibrary.convertTo18(addrBase, IERC20(addrBase).balanceOf(address(this))), balBaseLa
        );
        {
        uint256 sum1 = SafeMath.sub(
            SafeMath.div(
                SafeMath.mul(SafeMath.sub(usrMarginU, capPoolU), amtLeverTokenIn), leverTotalSupply
            ), feeUDelta);
        uint256 sum2 = SafeMath.div(SafeMath.mul(amtLeverTokenInValue, usrPoolQ), leverTotalSupply);
        uint256 sum3 = SafeMath.div(SafeMath.mul(amtLeverTokenInValue, underlyingQTYlast), leverTotal
        require((sum1 + sum2) > SafeMath.sub(sum3, swapU), "TeeterUnderlyingTOP:INSUFFICIENT AMOUNT U
        amtU = sum1 + sum2 - SafeMath.sub(sum3, swapU);
        }
        uint256 sub2 = SafeMath.div(SafeMath.mul(amtLeverTokenIn, usrMarginU), leverTotalSupply);
        usrMarginU = SafeMath.sub(usrMarginU, sub2);
        capU = SafeMath.add(capU, SafeMath.div(SafeMath.mul(capPoolU, amtLeverTokenIn), leverTotalSup
        capPoolU = SafeMath.sub(
            capPoolU,
            SafeMath.div(SafeMath.mul(capPoolU, amtLeverTokenIn), leverTotalSupply)
        );
        usrPoolQ = SafeMath.sub(usrPoolQ, SafeMath.div(SafeMath.mul(usrPoolQ, amtLeverTokenIn), lever
        fundSoldQTY = SafeMath.sub(fundSoldQTY, amtLeverTokenIn);
        fundSoldValueEn = SafeMath.mul(fundSoldQTY, nvEn);
        ITeeterLeverage(leverage).burn(address(this), amtLeverTokenIn);
        if(amtU != 0){
            balBaseLast = SafeMath.sub(SafeMath.add(balBaseLast, swapU), amtU);//update bal of base
            TransferHelper.safeTransfer(addrBase, to, TeeterLibrary.convert18ToOri(addrBase, amtU));
        }

    }

    function recycle(address to) external lock returns(uint256 amtToken0, uint256 amtU){
        require(status == 1, "TeeterUnderlyingTOP: FUNDCLOSE");
        updatePrice();
        uint256 amtLPT = this.balanceOf(address(this));
        require((amtLPT>0), "TeeterUnderlyingTOP: BALANCEERR");
        _updateIndexes();
        if(status != 1){TransferHelper.safeTransfer(address(this), to, amtLPT); return (0, 0);}
        uint256 hon = underlyingU + capU + (SafeMath.mul(feeU, SafeMath.sub(51922968585834827628530496
        uint256 ipjl = (SafeMath.mul(SafeMath.sub(underlyingQTY, usrPoolQ), priceEn)>>112) + capPoolU
        uint256 poolValue = hon + ipjl;
        uint256 amtMaxRecy = SafeMath.div(SafeMath.mul(totalSupply, hon), poolValue);
        if(amtLPT > amtMaxRecy){
            uint256 amtLPTReturn;
            amtLPTReturn = amtLPT - amtMaxRecy;//has judged before
            TransferHelper.safeTransfer(address(this), to, amtLPTReturn);
            amtLPT = amtMaxRecy;
        }
```

```
        amtU = SafeMath.div(SafeMath.mul(amtLPT, poolValue), totalSupply);
        ownerU += SafeMath.div(SafeMath.mul(SafeMath.mul(ownerRateEn, amtLPT)>>112, feeU), totalSuppl
        capU = SafeMath.sub(
            capU,
            SafeMath.div(SafeMath.mul(amtU, capU), hon)
        );
        feeU -= SafeMath.div(SafeMath.mul(feeU, amtLPT), totalSupply);//feeU > feeU*amtLPT/totalSuppl
        underlyingU = SafeMath.sub(underlyingU, SafeMath.div(SafeMath.mul(amtU, underlyingU), hon));
        _burn(address(this), amtLPT);
        if(amtU != 0){
            balBaseLast = SafeMath.sub(balBaseLast, amtU);//update bal of base
            capAddU = SafeMath.sub(capAddU, amtU);
            TransferHelper.safeTransfer(addrBase, to, TeeterLibrary.convert18ToOri(addrBase, amtU));
        }

    }

    function liquidation3Part(address to)external lock returns(uint256 amtToken0){
        require(status == 0, "TeeterUnderlyingTOP: FUNDOPEN");
        updatePrice();
        _updateIndexes();
        uint256 amtMaxLiqu = SafeMath.mul(underlyingValueEn>>112, liquDiscountRateEn)>>112;
        uint256 balanceBase = TeeterLibrary.convertTo18(addrBase, IERC20(addrBase).balanceOf(address(
        uint256 amtTokenIn = SafeMath.sub(balanceBase, balBaseLast);
        require(amtTokenIn > 0, "TeeterUnderlyingTOP: INSUFFICIENT_U_INPUT");//NO mix limit
        if(amtTokenIn > amtMaxLiqu){
            uint256 amtTokenInReturn = amtTokenIn - amtMaxLiqu;
            TransferHelper.safeTransfer(addrBase, to, TeeterLibrary.convert18ToOri(addrBase, amtToken
            balanceBase = SafeMath.sub(balanceBase, amtTokenInReturn);
            amtTokenIn = amtMaxLiqu;
        }
        amtToken0 = SafeMath.div(
            SafeMath.div(amtTokenIn<<112, liquDiscountRateEn)<<112,
            priceEn
        );
        underlyingQTY = SafeMath.sub(underlyingQTY, amtToken0);
        underlyingValueEn = SafeMath.mul(underlyingQTY, priceEn);
        balBaseLast = balanceBase;//update bal of base
        capU += amtTokenIn;
        if(amtToken0 !=0){
            TransferHelper.safeTransfer(token0, to, TeeterLibrary.convert18ToOri(token0, amtToken0));
        }

    }

    function liquidationLPT(address to)external lock returns(uint256 amtToken0, uint256 amtU){
        require((status == 0 || status == 2), "TeeterUnderlyingTOP: FUNDOPEN");
        uint256 amtTokenIn = this.balanceOf(address(this));
        require(amtTokenIn > 0, "TeeterUnderlyingTOP: INSUFFICIENT_LPTIN");
        amtToken0 = SafeMath.div(
            SafeMath.mul(underlyingQTY, amtTokenIn),
            totalSupply
        );
        underlyingQTY -= amtToken0;
        uint256 balBase = TeeterLibrary.convertTo18(addrBase, IERC20(addrBase).balanceOf(address(this
        amtU = SafeMath.div(
            SafeMath.mul(SafeMath.sub(balBase, ownerU), amtTokenIn),
            totalSupply
        );
        if(amtToken0 != 0){
            TransferHelper.safeTransfer(token0, to, TeeterLibrary.convert18ToOri(token0, amtToken0));
        }
        if(amtU != 0){
            balBaseLast = SafeMath.sub(balBaseLast, amtU);
            uint256 capUDelta = SafeMath.div(SafeMath.mul(amtU, capU), (capU+underlyingU));
            capU -= capUDelta;
```

```
            underlyingU =SafeMath.sub(underlyingU, SafeMath.sub(amtU,capUDelta));
            TransferHelper.safeTransfer(addrBase, to, TeeterLibrary.convert18ToOri(addrBase, amtU));
        }
        _burn(address(this), amtTokenIn);
    }

    function liquidationLT(address to)external lock returns(uint256 amtU){
        require(status == 2, "TeeterUnderlyingTOP: NOT FORCE CLOSE");
        uint256 amtLeverTokenIn = IERC20(leverage).balanceOf(address(this));
        require(amtLeverTokenIn >0, "TeeterUnderlyingTOP: balance0Err");
        uint256 leverTotalSupply = IERC20(leverage).totalSupply();
        amtU = SafeMath.div(SafeMath.mul(amtLeverTokenIn, usrMarginU), leverTotalSupply);
        usrMarginU -= amtU;
        if(amtU != 0){
            balBaseLast = SafeMath.sub(balBaseLast, amtU);
            TransferHelper.safeTransfer(addrBase, to, TeeterLibrary.convert18ToOri(addrBase, amtU));
        }
        ITeeterLeverage(leverage).burn(address(this), amtLeverTokenIn);
    }

    function cake(bool isTransfer)external returns(uint256 amtU){
        require((msg.sender == ITeeterFactory(factory).owner()) && ( ownerU > 0 || feeU > 0), "Teeter
        capU += (SafeMath.mul(feeU, SafeMath.sub(5192296858534827628530496329220096, ownerRateEn))>>1
        ownerU += (SafeMath.mul(feeU, ownerRateEn)>>112);
        amtU = ownerU;
        feeU = 0;
        if(isTransfer){
            balBaseLast = SafeMath.sub(balBaseLast, amtU);
            ownerU = 0;
            TransferHelper.safeTransfer(addrBase, ITeeterFactory(factory).owner(), TeeterLibrary.conv
        }
    }

    function closeForced()external returns(uint8 fundStatus){
        require(msg.sender == ITeeterFactory(factory).owner(), "TeeterUnderlying: FORBIDDEN");
        status = 2;
        uint256 feeUOwner = (SafeMath.mul(feeU, SafeMath.sub(5192296858534827628530496329220096, owne
        ownerU += feeUOwner;
        capU = SafeMath.sub(balBaseLast, (ownerU + underlyingU + usrMarginU));
        feeU = 0;
        fundSoldValueEn = 0;
        fundSoldQTY = 0;
        nvEn = 0;
        presLeverEn = 0;
        usrPoolQ = 0;
        capPoolU = 0;
        price0En = priceEn;
        fundStatus = status;
    }
}
```


Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires

the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unexpected Blockchain Currency

- **Description:**
  Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Entropy Illusion

- **Description:**
  All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

External Contract Referencing

- **Description:**
  One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Description:**
  Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default

types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Permission restrictions

- **Description:**
  Contract managers who can control liquidity or pledge pools, etc., or impose unreasonable restrictions on other users.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

armors.io

contact@armors.io