知道创宇

密级 ★ | 项目组公开

智能合约审计报告

安全状态

安全

★ ★ ★ ★ ★

主测人： Knownsec Blockchain Security Research

Release notes

Revise the	time	reviser	The version
Written document	20210115	Knownsec Blockchain Security Research Team	V1.0

Document information

The document name	Document	Report number	Level of
Ten Smart Contract Audit Report	V1.0	383115534a2e41af90a56f6d294a0bc0	Project team disclosure

The statement

Knownsec will only issue this report on the facts that have occurred or existed prior to the issuance of this report, and assume the corresponding responsibilities therefor.Knownsec cannot judge the security status of its smart contract and shall not be liable for the occurrence or existence of the facts after issuance.The security audit analysis and other contents in this report are only based on the documents and materials provided by the information provider to Knownsec as of the date of issuance of this report.Knownsec assumes that there is no missing, tampering, deletion or concealment of the information provided.If the information provided is missing, tampered with, deleted, concealed or reflected is inconsistent with the actual situation, Knownsec shall not be liable for any loss or adverse impact caused thereby.

directory

1. Review ..... - 6 -

2. Code vulnerability analysis ..... - 8 -

2.1 Vulnerability level distribution..... - 8 -

2.2 Summary of audit results..... - 9 -

3. Business security detection ..... - 12 -

3.1. Token function [approved] ..... - 12 -

3.2. Tenet Agreement Agreement [Approved] ..... - 20 -

3.3. Tenet Mining Information Contract [Approved] ..... - 26 -

3.4. Tenet Agent Contract [Approved] ..... - 29 -

3.5. Token conversion function [passed] ..... - 47 -

4. Basic vulnerability detection of code ..... - 56 -

4.1. Compiler version security [pass] ..... - 56 -

4.2. Redundant code [pass] ..... - 56 -

4.3. The use of secure arithmetic library [pass] ..... - 56 -

4.4. Not Recommended Coding Method [Passed] ..... - 57 -

4.5. Fair use of require/assert [pass] ..... - 57 -

4.6. Fallback function security [pass] ..... - 57 -

4.7.TX.Origin Authentication [Pass] ..... - 58 -

4.8. Owner permission control [pass] ..... - 58 -

4.9. Gas Consumption Detection [Passed] ..... - 58 -

4.10. Call injection attack [pass] ..... - 59 -

4.11. Low-level Function Security [Pass] ..... - 59 -

4.12. Additional Issue of Token Vulnerability [Pass] ..... - 59 -

4.13. Access control defect detection [pass] ..... - 60 -

4.14. Numerical overflow detection [pass] ..... - 60 -

4.15. Arithmetic Accuracy Error [Pass] ..... - 61 -

4.16. Incorrect use of random numbers [pass] ..... - 61 -

4.17. Unsecure interface use [pass] ..... - 62 -

4.18. Variable Overrides [Pass] ..... - 62 -

4.19. Uninitialized Stored Pointer [Pass] ..... - 62 -

4.20. Validation of return value call [Pass] ..... - 63 -

4.21. Transaction Order Depends on [Pass] ..... - 64 -

4.22. Timestamp Dependency Attacks [Pass] ..... - 64 -

4.23. Denial of Service Attacks [Pass] ..... - 65 -

4.24. False recharge vulnerability [pass] ..... - 65 -

4.25. Reentrant attack detection [Pass] ..... - 66 -

4.26. Replay Attack Detection [Pass] ..... - 66 -

4.27. Rearranged Attack Detection [Pass] ..... - 67 -

5. Appendix A: Contract Code ..... - 68 -

6. Appendix B: Safety risk rating criteria ..... - 97 -

7. Appendix C: Introduction to Smart Contract Security Audit Tools ..... - 98 -

6.1 Manticore ..... - 98 -

6.2 Oyente.....- 98 -

6.3 securify. Sh..... - 98 -

6.4 Echidna..... - 98 -

6.5 MAIAN.....- 99 -

6.6 ethersplay.....- 99 -

6.7 IDA - evm entry.....- 99 -

6.8 want - ide..... - 99 -

6.9 Know Chuang Yu Blockchain Security Auditor Special Toolkit.....- 99 -

1. review

This report will be valid for testing from January 14, 2021 to January 15, 2021, during which the security and compliance of the TEN smart contract code will be audited and used as the statistical basis for the report.

In this test, we know that Knownsec engineers have made a comprehensive analysis of the common vulnerabilities of smart contracts (see Chapter 3), and the comprehensive assessment is passed.

Result of this smart contract security audit: Passed

Since this test was conducted in a non-production environment, all the codes were the latest backup. During the test, the relevant interface personnel were communicated with, and the relevant test operations were carried out under the condition that the operational risks were controllable, so as to avoid the production and operation risks and code security risks during the test.

Report information of this audit:

Report number: 383115534 a2e41af90a56f6d294a0bc0

Report query address link:

https://attest.im/attestation/searchResult?qurey=383115534a2e41af90a56f6d294a0bc0

Objective information of this test:

entry	describe
The name of the Token	TENET(TEN)
Code type	Token code, DEFI protocol code
Code language	Solidity

Contract documents and hashes:

The contract documents	MD5

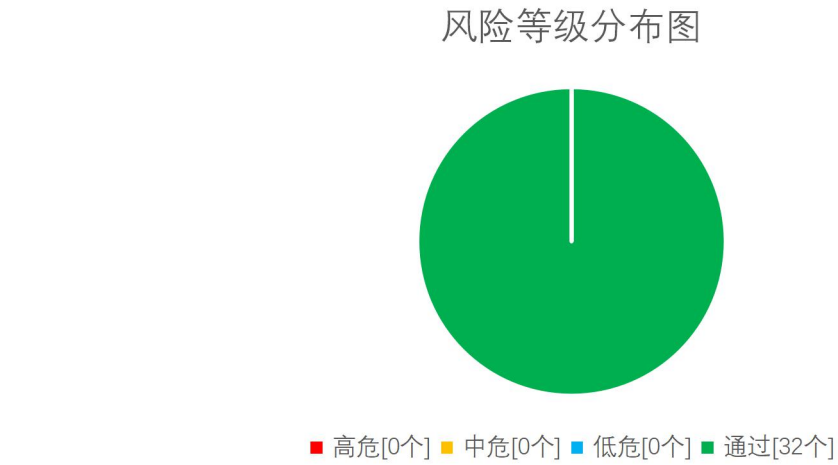
GovernorAlpha.sol	aeea1f656c7e8cd39d9830de6f7d6167
Migrations.sol	6f3dc9936ec2cff1666c76993e7d6531
MockERC20.sol	ec4d0e275520e7ba3ed3d0593134e298
Tenet.sol	3a71145c1611e71295f66e643e3a41aa
TenetMine.sol	0484003f13de11456ae068af1033d81b
TenetProxy.sol	4380cca859708e59fdf9513f0d234ec4
TenetProxyInner.sol	f910002792bd9907ea6d0a1435f70cfe
TenetSwap.sol	330974417c7c9813a781384a5073793d
TenetToken.sol	9224f099d482500dbf3cc1a0a3726bfd
Timelock.sol	4d6e6008ba8390000825d89f88a5b5ca

2. Code vulnerability analysis

2.1 Vulnerability level distribution

The vulnerability risk is counted according to the level:

Statistical table of the number of security risk levels			
At high risk of	In a crisis	Low risk	through
0	0	0	32



2.2 Summary of audit results

The audit results			
The audit project	The audit content	state	describe
Business security detection	Scrip function	through	After testing, there is no safety problem.
	Tenet Agreement Contract	through	After testing, there is no safety problem.
	Tenet Mining Information Contract	through	After testing, there is no safety problem.
	Tenet agent class contract	through	After testing, there is no safety problem.
	Token conversion function	through	After testing, there is no safety problem.
Basic code vulnerability detection	Compiler versioning security	through	After testing, there is no such safety problem.
	Redundant code	through	After testing, there is no such safety problem.
	Use of secure arithmetic libraries	through	After testing, there is no such safety problem.
	Not recommended for coding	through	After testing, there is no such safety problem.
	Fair use of require/assert	through	After testing, there is no such safety problem.
	The fallback	through	After testing, there is no such safety

	function is secure		problem.
	Tx.org in authentication	through	After testing, there is no such safety problem.
	Owner permission control	through	After testing, there is no such safety problem.
	Gas Consumption Detection	through	After testing, there is no such safety problem.
	Call injection attack	through	After testing, there is no such safety problem.
	Low-level Function Security	through	After testing, there is no such safety problem.
	Added token vulnerability	through	After testing, there is no such safety problem.
	Access control defect detection	through	After testing, there is no such safety problem.
	Numerical overflow detection	through	After testing, there is no such safety problem.
	Error in arithmetic accuracy	through	After testing, there is no such safety problem.
	Error using random number detection	through	After testing, there is no such safety problem.
	Unsecure interface use	through	After testing, there is no such safety problem.
	Variables are covered	through	After testing, there is no such safety problem.
	Uninitialized storage pointer	through	After testing, there is no such safety problem.
	The return value	through	After testing, there is no such safety

	invokes validation		problem.
	Transaction order dependence detection	through	After testing, there is no such safety problem.
	The timestamp depends on the attack	through	After testing, there is no such safety problem.
	Denial of service attack detection	through	After testing, there is no such safety problem.
	Detection of false recharge vulnerability	through	After testing, there is no such safety problem.
	Reentrant attack detection	through	After testing, there is no such safety problem.
	Replay attack detection	through	After testing, there is no such safety problem.
	Rearrange attack detection	through	After testing, there is no such safety problem.

3. Business security detection

3.1. Token function [pass]

Audit analysis: The token function is implemented in the tenettoken.sol contract. It is ERC20 compliant and can be used for community governance voting.

```
contract TenetToken is ERC20("Tenet", "TEN"), Ownable {
    // @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
    function mint(address _to, uint256 _amount) public onlyOwner {
        _mint(_to, _amount);
        _moveDelegates(address(0), _delegates[_to], _amount);
    }
    function burn(address _account, uint256 _amount) public onlyOwner {
        _burn(_account, _amount);
    }
    // Copied and modified from YAM code:
    // https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAMGovernanceStorage.sol
    // https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAMGovernance.sol
    // Which is copied and modified from COMPOUND:
    // https://github.com/compound-finance/compound-protocol/blob/master/contracts/Governance/Comp.sol

    // @notice A record of each accounts delegate
    mapping (address => address) internal _delegates;
```

```
    // @notice A checkpoint for marking number of votes from a given block
    struct Checkpoint {
        uint32 fromBlock;
        uint256 votes;
    }

    // @notice A record of votes checkpoints for each account, by index
    mapping (address => mapping (uint32 => Checkpoint)) public checkpoints;

    // @notice The number of checkpoints for each account
    mapping (address => uint32) public numCheckpoints;

    // @notice The EIP-712 typehash for the contract's domain
    bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");

    // @notice The EIP-712 typehash for the delegation struct used by the contract
    bytes32 public constant DELEGATION_TYPEHASH = keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)");

    // @notice A record of states for signing / validating signatures
    mapping (address => uint) public nonces;

    // @notice An event thats emitted when an account changes its delegate
    event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);

    // @notice An event thats emitted when a delegate account's vote balance changes
    event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);

    /**
     * @notice Delegate votes from `msg.sender` to `delegatee`
```

```
    * @param delegator The address to get delegatee for
    */
  function delegates(address delegator)
        external
        view
        returns (address)
  {
        return _delegates[delegator];
  }

 /**
  * @notice Delegate votes from `msg.sender` to `delegatee`
  * @param delegatee The address to delegate votes to
  */
  function delegate(address delegatee) external {
        return _delegate(msg.sender, delegatee);
  }

  /**
    * @notice Delegates votes from signatory to `delegatee`
    * @param delegatee The address to delegate votes to
    * @param nonce The contract state required to match the signature
    * @param expiry The time at which to expire the signature
    * @param v The recovery byte of the signature
    * @param r Half of the ECDSA signature pair
    * @param s Half of the ECDSA signature pair
    */
  Function DelegateBySig (// knownSec
        address delegatee,
        uint nonce,
        uint expiry,
        uint8 v,
        bytes32 r,
```

```
        bytes32 s
    )
        external
    {
        bytes32 domainSeparator = keccak256(
            abi.encode(
                DOMAIN_TYPEHASH,
                keccak256(bytes(name())),
                getChainId(),
                address(this)
            )
        );

        bytes32 structHash = keccak256(
            abi.encode(
                DELEGATION_TYPEHASH,
                delegatee,
                nonce,
                expiry
            )
        );

        bytes32 digest = keccak256(
            abi.encodePacked(
                "\x19\x01",
                domainSeparator,
                structHash
            )
        );

        address signatory = ecrecover(digest, v, r, s);
        require(signatory ! = address(0), "delegateBySig: invalid signature");
        require(nonce == nonces[signatory]++, "delegateBySig: invalid nonce");
```

```
        require(now <= expiry, "delegateBySig: signature expired");
        return _delegate(signatory, delegatee);
    }

    /**
     * @notice Gets the current votes balance for `account`
     * @param account The address to get votes balance
     * @return The number of current votes for `account`
     */
    function getCurrentVotes(address account)
        external
        view
        returns (uint256)
    {/// knownSec // Get the current vote
        uint32 nCheckpoints = numCheckpoints[account];
        return nCheckpoints > 0 ?      checkpoints[account][nCheckpoints - 1].votes : 0;
    }

    /**
     * @notice Determine the prior number of votes for an account as of a block number
     * @dev Block number must be a finalized block or else this function will revert to prevent misinformation.
     * @param account The address of the account to check
     * @param blockNumber The block number to get the vote balance at
     * @return The number of votes the account had as of the given block
     */
    function getPriorVotes(address account, uint blockNumber)
        external
        view
        returns (uint256)
    {/// knownSec // Get the previous vote
        require(blockNumber < block.number, "getPriorVotes: not yet determined");
```

```
        uint32 nCheckpoints = numCheckpoints[account];
        if (nCheckpoints == 0) {
            return 0;
        }

        // First check most recent balance
        if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
            return checkpoints[account][nCheckpoints - 1].votes;
        }

        // Next check implicit zero balance
        if (checkpoints[account][0].fromBlock > blockNumber) {
            return 0;
        }

        uint32 lower = 0;
        uint32 upper = nCheckpoints - 1;
        while (upper > lower) {
            uint32 center = upper - (upper - lower) / 2;      // ceil, avoiding overflow
            Checkpoint memory cp = checkpoints[account][center];
            if (cp.fromBlock == blockNumber) {
                return cp.votes;
            } else if (cp.fromBlock < blockNumber) {
                lower = center;
            } else {
                upper = center - 1;
            }
        }
        return checkpoints[account][lower].votes;
    }

    function _delegate(address delegator, address delegatee)
        internal
```

```
    {
        address currentDelegate = _delegates[delegator];
        uint256 delegatorBalance = balanceOf(delegator);      // balance of underlying TNTs (not scaled);
        _delegates[delegator] = delegatee;

        emit DelegateChanged(delegator, currentDelegate, delegatee);

        _moveDelegates(currentDelegate, delegatee, delegatorBalance);
    }

    function _moveDelegates(address srcRep, address dstRep, uint256 amount) internal {
        if (srcRep != dstRep && amount > 0) {
            if (srcRep != address(0)) {
                // decrease old representative
                uint32 srcRepNum = numCheckpoints[srcRep];
                uint256 srcRepOld = srcRepNum > 0 ?      checkpoints[srcRep][srcRepNum - 1].votes : 0;
                uint256 srcRepNew = srcRepOld.sub(amount);
                _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
            }

            if (dstRep != address(0)) {
                // increase new representative
                uint32 dstRepNum = numCheckpoints[dstRep];
                uint256 dstRepOld = dstRepNum > 0 ?      checkpoints[dstRep][dstRepNum - 1].votes : 0;
                uint256 dstRepNew = dstRepOld.add(amount);
                _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
            }
        }
    }
```

```
    function _writeCheckpoint(
        address delegatee,
        uint32 nCheckpoints,
        uint256 oldVotes,
        uint256 newVotes
    )
        internal
    {
        uint32 blockNumber = safe32(block.number, "_writeCheckpoint: block number exceeds 32 bits");

        if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
            checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
        } else {
            checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
            numCheckpoints[delegatee] = nCheckpoints + 1;
        }

        emit DelegateVotesChanged(delegatee, oldVotes, newVotes);
    }

    function safe32(uint n, string memory errorMessage) internal pure returns (uint32) {
        require(n < 2**32, errorMessage);
        return uint32(n);
    }

    Function getChainId() internal pure returns (uint) {// knownSec //
        uint256 chainId;
        assembly { chainId := chainid() }
        return chainId;
    }
```

```
}
```

Safety advice: none.

3.2. Tenet Agreement Agreement [Approved]

Audit Analysis: The Tenet protocol functions are implemented in the tenet.sol contract and are primarily used to provide the core functions of liquid mining.

```
contract Tenet is Ownable {
    .
    constructor(
        TenetToken _ten,
        TenetMine _tenMineCalc,
        IERC20 _lpTen,
        address _devaddr,
        uint256 _allocPointProject,
        uint256 _allocPointUser,
        uint256 _devWithdrawStartBlock,
        uint256 _modifyAllocPointPeriod,
        uint256 _bonusAllocPointBlock,
        uint256 _minProjectUserCount
    ) public {
        ten = _ten;
        tenMineCalc = _tenMineCalc;
        devaddr = _devaddr;
        lpTokenTen = _lpTen;
        tenProjectPool.allocPoint = _allocPointProject;
        tenUserPool.allocPoint = _allocPointUser;
        totalAllocPoint = _allocPointProject.add(_allocPointUser);
        devaddrAmount = 0;
        devWithdrawStartBlock = _devWithdrawStartBlock;
```

```
        addpoolfee = 0;
        emergencyWithdraw = 0;
        modifyAllocPointPeriod = _modifyAllocPointPeriod;
        lastModifyAllocPointBlock = tenMineCalc.startBlock();
        bonusAllocPointBlock = _bonusAllocPointBlock;
        minProjectUserCount = _minProjectUserCount;
        bonusAddr = msg.sender;
        bonusPoolFeeRate = 0;
    }
    .
    function add(address _lpToken,
            address _tokenAddr,
            uint256 _tokenAmount,
            uint256 _startBlock,
            uint256 _endBlockOffset,
            uint256 _tokenPerBlock,
            uint256 _tokenBonusEndBlockOffset,
            uint256 _tokenBonusMultipler,
            Uint256 _lpTenAmount) public onlyNotEmergencyWithdraw {// knownSec // Add
liquid mine pool
        if(_startBlock == 0){
            _startBlock = block.number;
        }
        require(block.number <= _startBlock, "add: startBlock invalid");
        require(_endBlockOffset >= _tokenBonusEndBlockOffset, "add: bonusEndBlockOffset
invalid");

require(tenMineCalc.getMultiplier(_startBlock,_startBlock.add(_endBlockOffset),_startBlock.add
(_endBlockOffset),_startBlock.add(_tokenBonusEndBlockOffset),_tokenBonusMultipler).mul(_tok
enPerBlock) <= _tokenAmount, "add: token amount invalid");
        require(_tokenAmount > 0, "add: tokenAmount invalid");
        IERC20(_tokenAddr).safeTransferFrom(msg.sender,address(this), _tokenAmount);
        if(addpoolfee > 0){
```

```
                IERC20(address(ten)).safeTransferFrom(msg.sender,address(this), addpoolfee);
                if(bonusPoolFeeRate > 0){
                    IERC20(address(ten)).safeTransfer(bonusAddr,
addpoolfee.mul(bonusPoolFeeRate).div(10000));
                    if(bonusPoolFeeRate < 10000){

ten.burn(address(this),addpoolfee.sub(addpoolfee.mul(bonusPoolFeeRate).div(10000)));
                    }
                }else{
                    ten.burn(address(this),addpoolfee);
                }
            }
            uint256 pid = poolInfo.length;
            PoolSettingInfo. Push (PoolSettingInfo({// knownSec // Add mine pool Settings
                lpToken: _lpToken,
                tokenAddr: _tokenAddr,
                projectAddr: msg.sender,
                tokenAmount: _tokenAmount,
                startBlock: _startBlock,
                endBlock: _startBlock.add(_endBlockOffset),
                tokenPerBlock: _tokenPerBlock,
                tokenBonusEndBlock: _startBlock.add(_tokenBonusEndBlockOffset),
                tokenBonusMultipler: _tokenBonusMultipler
            }));
            PoolInfo.push (poolInfo ({// knownSec // Add mine pool information
                lastRewardBlock: block.number > _startBlock ?      block.number : _startBlock,
                accTokenPerShare: 0,
                accTenPerShare: 0,
                lpTokenTotalAmount: 0,
                userCount: 0,
                amount: 0,
                rewardTenDebt: 0,
                mineTokenAmount: 0
```

```
        }));
        if(_lpTenAmount>=MINLPTOKEN_AMOUNT){
            depositTenByProject(pid,_lpTenAmount);
        }
        emit AddPool(msg.sender, pid, _tokenAmount,_lpTenAmount);
    }
    .
    Function DepositEnByProject (UINT256_PID, UINT256_Amount) public
OnlyNotemergencywithdraw {/// DepositEnByProject (UINT256_PID, UINT256_Amount
        require(_amount > 0, "depositTenByProject: lpamount not good");
        PoolInfo storage pool = poolInfo[_pid];
        PoolSettingInfo storage poolSetting = poolSettingInfo[_pid];
        require(poolSetting.projectAddr == msg.sender, "depositTenByProject: not good");
        _minePoolTen(tenProjectPool);
        _withdrawProjectTenPool(pool);
        _updateProjectTenPoolAmount(pool,_amount,1);
        Emit DepositLPTen (MSG. The sender, 1, _amount, 0, 0).
    }

    A function descriptive TenbyProject (UINT256_PID, UINT256_Amount) public {// knownSec
// Specifies a pool extract TEN
        PoolInfo storage pool = poolInfo[_pid];
        PoolSettingInfo storage poolSetting = poolSettingInfo[_pid];
        require(poolSetting.projectAddr == msg.sender, "withdrawTenByProject: not good");
        require(pool.amount >= _amount, "withdrawTenByProject: not good");
        if(emergencyWithdraw == 0){
            _minePoolTen(tenProjectPool);
            _withdrawProjectTenPool(pool);
        }else{
            if(pool.lastRewardBlock < poolSetting.endBlock){
                if(poolSetting.tokenAmount.sub(pool.mineTokenAmount) > 0){

IERC20(poolSetting.tokenAddr).safeTransfer(poolSetting.projectAddr,poolSetting.tokenAmount.su
```

```
b(pool.mineTokenAmount));
                    }
                }
            }
            if(_amount > 0){
                _updateProjectTenPoolAmount(pool,_amount,2);
            }
            Emit WithdrawLPTen (MSG. The sender, 1, _amount, 0, 0).
    }
    .
    Function depositLPToken(uint256_pid, uint256_amount) public onlyNotEmergencyWithdraw
{///knownsc// DepositLPToken (uint256_pid, uint256_amount) public onlyNotEmergencyWithdraw
{///knownsc//
            require(_amount > 0, "depositLPToken: lpamount not good");
            PoolInfo storage pool = poolInfo[_pid];
            PoolSettingInfo storage poolSetting = poolSettingInfo[_pid];
            UserInfo storage user = userInfo[_pid][msg.sender];
            _minePoolToken(pool,poolSetting);
            (uint256 pendingToken,uint256 pendingTen,uint256 freezeBlocks,uint256 freezeTen) =
_withdrawTokenPool(msg.sender,pool,user,poolSetting);
            if (user.amount < MINLPTOKEN_AMOUNT) {
                pool.userCount = pool.userCount.add(1);
            }
            IERC20(poolSetting.lpToken).safeTransferFrom(address(msg.sender),     address(this),
_amount);
            pool.lpTokenTotalAmount = pool.lpTokenTotalAmount.add(_amount);

_updateTokenPoolUser(pool.accTokenPerShare,pool.accTenPerShare,user,freezeBlocks,freezeTen,
_amount,1);
            emit                              Deposit(msg.sender,                              _pid,
_amount,pendingToken,pendingTen,freezeTen,freezeBlocks);
    }
```

```
A function withdrawing lpToken (uint256 _pid, uint256 _amount) public {// knownSec // Extract the liquid token from the specified pool
        require(_amount > 0, "withdrawLPToken: lpamount not good");
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        PoolSettingInfo storage poolSetting = poolSettingInfo[_pid];
        require(user.amount >= _amount, "withdrawLPToken: not good");
        if(emergencyWithdraw == 0){
            _minePoolToken(pool,poolSetting);
            (uint256 pendingToken,uint256 pendingTen,uint256 freezeBlocks,uint256 freezeTen) = _withdrawTokenPool(msg.sender,pool,user,poolSetting);

_updateTokenPoolUser(pool.accTokenPerShare,pool.accTenPerShare,user,freezeBlocks,freezeTen,_amount,2);
            emit Withdraw(msg.sender, _pid, _amount,pendingToken,pendingTen,freezeTen,freezeBlocks);
        }else{

_updateTokenPoolUser(pool.accTokenPerShare,pool.accTenPerShare,user,user.freezeBlocks,user.freezeTen,_amount,2);
            Emit Withdraw (MSG) sender, _pid, _amount, 0, 0, user. FreezeTen, user. FreezeBlocks);
        }
        IERC20(poolSetting.lpToken).safeTransfer(address(msg.sender), _amount);
        pool.lpTokenTotalAmount = pool.lpTokenTotalAmount.sub(_amount);
        if(user.amount < MINLPTOKEN_AMOUNT){
            pool.userCount = pool.userCount.sub(1);
        }
    }
    .
}
```

Safety advice: none.

3.3. Tenet Mining Information Contract [Approved]

Audit analysis: Tenet mining information contract function is implemented in tenetmine. sol contract, which is mainly used to query current mining information and calculate mining reward, etc.

```
contract TenetMine is Ownable {
    using SafeMath for uint256;
    struct MinePeriodInfo {
        uint256 tenPerBlockPeriod;
        uint256 totalTenPeriod;
    }
    uint256 public bonusEndBlock;
    uint256 public bonus_multiplier;
    uint256 public bonusTenPerBlock;
    uint256 public startBlock;
    uint256 public endBlock;
    uint256 public subBlockNumerPeriod;
    uint256 public totalSupply;
    MinePeriodInfo[] public allMinePeriodInfo;

    constructor(
        uint256 _startBlock,
        uint256 _bonusEndBlockOffset,
        uint256 _bonus_multiplier,
        uint256 _bonusTenPerBlock,
        uint256 _subBlockNumerPeriod,
        uint256[] memory _tenPerBlockPeriod
    ) public {
        startBlock = _startBlock>0 ?    _startBlock : block.number + 1;
        bonusEndBlock = startBlock.add(_bonusEndBlockOffset);
        bonus_multiplier = _bonus_multiplier;
```

```
        bonusTenPerBlock = _bonusTenPerBlock;
        subBlockNumerPeriod = _subBlockNumerPeriod;
        totalSupply =
bonusEndBlock.sub(startBlock).mul(bonusTenPerBlock).mul(bonus_multiplier);
        for (uint256 i = 0;  i < _tenPerBlockPeriod.length;  i++) {
            allMinePeriodInfo.push(MinePeriodInfo({
                tenPerBlockPeriod: _tenPerBlockPeriod[i],
                totalTenPeriod: totalSupply
            }));
            totalSupply =
totalSupply.add(subBlockNumerPeriod.mul(_tenPerBlockPeriod[i]));
        }
        endBlock =
bonusEndBlock.add(subBlockNumerPeriod.mul(_tenPerBlockPeriod.length));
    }
    Function set_startBlock(uint256 _startBlock) public onlyOwner {// knownSec //
set_startBlock
        require(block.number < _startBlock, "set_startBlock: startBlock invalid");
        uint256 bonusEndBlockOffset = bonusEndBlock.sub(startBlock);
        startBlock = _startBlock>0 ?  _startBlock : block.number + 1;
        bonusEndBlock = startBlock.add(bonusEndBlockOffset);
        endBlock =
bonusEndBlock.add(subBlockNumerPeriod.mul(allMinePeriodInfo.length));
    }
    function getMinePeriodCount() public view returns (uint256) {
        return allMinePeriodInfo.length;
    }
    Function calcminetenReply (uint256_from, uint256_to) public view returns (uint256) {//
knownSec returns (uint256_to
        if(_from < startBlock){
            _from = startBlock;
        }
        if(_from >= endBlock){
```

```
            return 0;
        }
        if(_from >= _to){
            return 0;
        }
        uint256 mineFrom = calcTotalMine(_from);
        uint256 mineTo= calcTotalMine(_to);
        return mineTo.sub(mineFrom);
    }
    Function calcTotalMine(uint256 _to) public view returns (uint256) {// knownSec returns ()
        if(_to <= startBlock){
            totalMine = 0;
        }else if(_to <= bonusEndBlock){
            totalMine = _to.sub(startBlock).mul(bonusTenPerBlock).mul(bonus_multiplier);
        }else if(_to < endBlock){
            uint256 periodIndex = _to.sub(bonusEndBlock).div(subBlockNumerPeriod);
            uint256 periodBlock = _to.sub(bonusEndBlock).mod(subBlockNumerPeriod);
            MinePeriodInfo memory minePeriodInfo = allMinePeriodInfo[periodIndex];
            uint256 curMine = periodBlock.mul(minePeriodInfo.tenPerBlockPeriod);
            totalMine = curMine.add(minePeriodInfo.totalTenPeriod);
        }else{
            totalMine = totalSupply;
        }
    }
    // Return reward multiplier over the given _from to _to block.
    Function getMultiplier (uint256 _from, uint256 _to, uint256 _end, uint256
_tokenBonusEndBlock, uint256 _tokenBonusMultipler) public pure returns (uint256) {// /
knownsec calculate according to the number of blocks / / factors
        if(_to > _end){
            _to = _end;
        }
        if(_from> _end){
            return 0;
```

```
            }else if (_to <= _tokenBonusEndBlock) {
                return _to.sub(_from).mul(_tokenBonusMultipler);
            } else if (_from >= _tokenBonusEndBlock) {
                return _to.sub(_from);
            } else {
                return
_tokenBonusEndBlock.sub(_from).mul(_tokenBonusMultipler).add(_to.sub(_tokenBonusEndBlock)
);
            }
        }
}
```

Safety advice: none.

3.4. Tenet Agent Contract [Approved]

Audit analysis: the function of Tenet proxy contract is implemented in TenetProxy.sol and TenetProxyInner.sol contract, which is mainly used to query the data information of various pools in Tenet protocol and the price of various tokens.

```
contract TenetProxy is Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    uint256 public constant MINLPTOKEN_AMOUNT = 10;
    uint public constant MINIMUM_LIQUIDITY = 10**3;
    uint256 public constant PERSHARERATE = 1000000000000;

    Tenet public tenet;
    TenetMine public tenetmine;
    constructor(Tenet _tenet) public {
        tenet = _tenet;
```

```
        tenetmine = tenet.tenMineCalc();
    }
    function set_tenet(Tenet _tenet) public onlyOwner {
        tenet = _tenet;
        tenetmine = tenet.tenMineCalc();
    }
    function getPoolAllInfo(uint256 _pid) public view returns (address[3] memory retData1,uint256[6] memory retData2,uint256[8] memory retData3) {
        (retData1) = getPoolSettingInfo1(_pid);
        (retData2) = getPoolSettingInfo2(_pid);
        (retData3) = getPoolInfo(_pid);
    }
    function getPoolSettingInfo1(uint256 _pid) public view returns (address[3] memory retData1) {
        (retData1[0],retData1[1],retData1[2],,,,,,) = tenet.poolSettingInfo(_pid);
    }
    function getPoolSettingInfo2(uint256 _pid) public view returns (uint256[6] memory retData2) {
        (,,,retData2[0],retData2[1],retData2[2],retData2[3],retData2[4],retData2[5]) = tenet.poolSettingInfo(_pid);
    }
    function getPoolInfo(uint256 _pid) public view returns (uint256[8] memory retData3) {
        (retData3[0],retData3[1],retData3[2],retData3[3],retData3[4],retData3[5],retData3[6],retData3[7]) = tenet.poolInfo(_pid);
    }

    function getPendingTenByProject(uint _pid) public view returns (uint256) {
        ( , ,uint256[8] memory retData3) = getPoolAllInfo(_pid);
        if(retData3[1] < MINLPTOKEN_AMOUNT){
            return 0;
        }
        if(retData3[5] < MINLPTOKEN_AMOUNT){
```

```
            return 0;
        }
        uint256[4] memory tenPoolInfo;
        (tenPoolInfo[0],tenPoolInfo[1],tenPoolInfo[2],tenPoolInfo[3]) =
tenet.tenProjectPool();
        if(tenPoolInfo[3] < MINLPTOKEN_AMOUNT){
            return 0;
        }
        if (block.number > tenPoolInfo[0] && retData3[5] ! = 0) {
            uint256 tenReward = tenetmine.calcMineTenReward(tenPoolInfo[0],
block.number);
            tenReward = tenReward.mul(tenPoolInfo[2]).div(tenet.totalAllocPoint());
            tenPoolInfo[1] = tenPoolInfo[1].add(tenReward.mul(1e12).div(tenPoolInfo[3]));
        }
        return retData3[5].mul(tenPoolInfo[1]).div(1e12).sub(retData3[6]);
    }
    function _calcFreezeTen(uint256[6] memory userInfo,uint256 accTenPerShare) internal
view returns (uint256 pendingTen,uint256 freezeBlocks,uint256 freezeTen){
        pendingTen =
userInfo[0].mul(accTenPerShare).div(PERSHARERATE).sub(userInfo[2]);
        uint256 blockNow = block.number.sub(userInfo[3]);
        uint256 periodBlockNumer = tenetmine.subBlockNumerPeriod();
        freezeBlocks = blockNow.add(userInfo[4]);
        if(freezeBlocks <= periodBlockNumer){
            freezeTen = pendingTen.add(userInfo[5]);
            pendingTen = 0;
        }else{
            if(pendingTen == 0){
                freezeBlocks = 0;
                freezeTen = 0;
                pendingTen = userInfo[5];
            }else{
                freezeTen =
```

```
pendingTen.add(userInfo[5]).mul(periodBlockNumer).div(freezeBlocks);
                    pendingTen = pendingTen.add(userInfo[5]).sub(freezeTen);
                    freezeBlocks = periodBlockNumer;
                }
            }
        }

        function getPendingTenByUser(address _user) public view returns (uint256,uint256,uint256)
{
            uint256[6] memory userInfo;
            (userInfo[0],userInfo[1],userInfo[2],userInfo[3],userInfo[4],userInfo[5])  =
tenet.userInfoUserPool(_user);
            if(userInfo[0] < MINLPTOKEN_AMOUNT){
                if(block.number.sub(userInfo[3])>tenetmine.subBlockNumerPeriod()){
                    Return (the userInfo [5], 0, 0).
                }else{
                    Return (0, 0, the userInfo [5]);
                }
            }
            uint256[4] memory tenPoolInfo;
            (tenPoolInfo[0],tenPoolInfo[1],tenPoolInfo[2],tenPoolInfo[3]) = tenet.tenUserPool();
            if(tenPoolInfo[3] < MINLPTOKEN_AMOUNT){
                if(block.number.sub(userInfo[3])>tenetmine.subBlockNumerPeriod()){
                    Return (the userInfo [5], 0, 0).
                }else{
                    Return (0, 0, the userInfo [5]);
                }
            }
            if (block.number > tenPoolInfo[0]) {
                uint256 tenReward = tenetmine.calcMineTenReward(tenPoolInfo[0],
block.number);
                tenReward = tenReward.mul(tenPoolInfo[2]).div(tenet.totalAllocPoint());
                tenPoolInfo[1] = tenPoolInfo[1].add(tenReward.mul(1e12).div(tenPoolInfo[3]));
```

```
        }
        (uint256 pendingTen,uint256 freezeBlocks,uint256 freezeTen) =
_calcFreezeTen(userInfo,tenPoolInfo[1]);
        return (pendingTen,freezeBlocks,freezeTen);
    }


    Function getPendingTen(uint256,uint256,uint256) public view returns
(uint256,uint256,uint256) {//knownsec// getPendingTen
        uint256[6] memory userInfo;
        (userInfo[0], ,userInfo[2],userInfo[3],userInfo[4],userInfo[5]) =
tenet.userInfo(_pid,_user);
        if(userInfo[0] < MINLPTOKEN_AMOUNT){
            if(block.number.sub(userInfo[3])>tenetmine.subBlockNumerPeriod()){
                Return (the userInfo [5], 0, 0).
            }else{
                Return (0, 0, the userInfo [5]);
            }
        }
        ( , ,uint256[8] memory retData3) = getPoolAllInfo(_pid);
        if(retData3[1] < MINLPTOKEN_AMOUNT){
            if(block.number.sub(userInfo[3])>tenetmine.subBlockNumerPeriod()){
                Return (the userInfo [5], 0, 0).
            }else{
                Return (0, 0, the userInfo [5]);
            }
        }
        uint256 pending = getPendingTenByProject(_pid);
        retData3[3] = retData3[3].add(pending.mul(1e12).div(retData3[1]));
        (uint256 pendingTen,uint256 freezeBlocks,uint256 freezeTen) =
_calcFreezeTen(userInfo,retData3[3]);
        return (pendingTen,freezeBlocks,freezeTen);
    }
```

```
    Function getPendingToken(uint256 _pid, address _user) public view returns (uint256) {/// knownSec // Get the pending token
        ( ,uint256[6] memory retData2,uint256[8] memory retData3) = getPoolAllInfo(_pid);
        if(retData3[1] < MINLPTOKEN_AMOUNT){
            return 0;
        }
        uint256[6] memory userInfo;
        (userInfo[0],userInfo[2], , , , ) = tenet.userInfo(_pid,_user);
        if(userInfo[0] < MINLPTOKEN_AMOUNT){
            return 0;
        }
        if (block.number > retData3[0] && retData3[1] != 0) {
            uint256 tokenReward = retData2[3].mul(tenetmine.getMultiplier(retData3[0], block.number,retData2[2],retData2[4],retData2[5]));
            retData3[2] = retData3[2].add(tokenReward.mul(1e12).div(retData3[1]));
        }
        return userInfo[0].mul(retData3[2]).div(1e12).sub(userInfo[2]);
    }
    Function calcLiquidity2(address _pairAddr,uint256 _token0Amount,uint256 _token1Amount) public liquidity = function calcLiquidity2(address _pairAddr,uint256 _token0Amount,uint256 _token1Amount
        uint256 totalSupply = IUniswapV2Pair(_pairAddr).totalSupply();
        (uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(_pairAddr).getReserves();
        if(totalSupply == 0){
            liquidity = sqrt(_token0Amount.mul(_token1Amount)).sub(MINIMUM_LIQUIDITY);
        }else {
            liquidity = min(_token0Amount.mul(totalSupply) / reserve0, _token1Amount.mul(totalSupply) / reserve1);
        }
    }
```

```
Function calcLiquidity(address _pairAddr,address _tokenAddr,uint256 _tokenAmount) public view returns (uint256 liquidity) {// knownSec // Calculate the liquidity
        uint256[2] memory tokenAmountOut;
        if(_tokenAddr == IUniswapV2Pair(_pairAddr).token0()){
            (tokenAmountOut[0],tokenAmountOut[1]) = calcTokenXOut(_pairAddr,_tokenAddr,_tokenAmount,0);
        }else if(_tokenAddr == IUniswapV2Pair(_pairAddr).token1()){
            (tokenAmountOut[0],tokenAmountOut[1]) = calcTokenXOut(_pairAddr,_tokenAddr,_tokenAmount,1);
        }else{
            (tokenAmountOut[0],tokenAmountOut[1]) = calcTokensOut(_pairAddr,_tokenAddr,_tokenAmount);
        }
        if(tokenAmountOut[0] == 0){
            liquidity = 0;
        }else if(tokenAmountOut[0] == 0){
            liquidity = 0;
        }else{
            uint256 totalSupply = IUniswapV2Pair(_pairAddr).totalSupply();
            (uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(_pairAddr).getReserves();
            if(totalSupply == 0){
                liquidity = sqrt(tokenAmountOut[0].mul(tokenAmountOut[1])).sub(MINIMUM_LIQUIDITY);
            }else {
                liquidity = min(tokenAmountOut[0].mul(totalSupply) / reserve0, tokenAmountOut[1].mul(totalSupply) / reserve1);
            }
        }
    }

    function getAmountOut(address _pairAddr, address _fromAddr,uint amountIn) public view virtual returns (uint256){
```

```
        //require(amountIn > 0, 'getAmountOut: INSUFFICIENT_INPUT_AMOUNT');
        if(amountIn == 0){
            return 0;
        }
        (uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(_pairAddr).getReserves();
        //require(reserve0 > 0 && reserve1 > 0, 'getAmountOut: INSUFFICIENT_LIQUIDITY');
        if(reserve0 == 0){
            return 0;
        }
        if(reserve1 == 0){
            return 0;
        }
        uint amountInWithFee = amountIn.mul(997);
        if(_fromAddr == IUniswapV2Pair(_pairAddr).token0()){
            uint numerator = amountInWithFee.mul(reserve1);
            uint denominator = reserve0.mul(1000).add(amountInWithFee);
            return numerator.div(denominator);
        }else{
            uint numerator = amountInWithFee.mul(reserve0);
            uint denominator = reserve1.mul(1000).add(amountInWithFee);
            return numerator.div(denominator);
        }
    }

    Function getPrice(address _pairAddr, address _fromAddr) public view returns (uint256) {// knownSec //
        (uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(_pairAddr).getReserves();
        if(_fromAddr == IUniswapV2Pair(_pairAddr).token0()){
            return reserve1.mul(1e12).div(reserve0);
        }else{
            return reserve0.mul(1e12).div(reserve1);
        }
```

```
    }

    function calcTokensOut(address _pairAddr,address _tokenAddr,uint256 _tokenAmount) public view returns (uint256,uint256) {
        IUniswapV2Factory factory = IUniswapV2Factory(IUniswapV2Pair(_pairAddr).factory());
        //require(address(factory) != address(0), 'calcTokensOut: INSUFFICIENT_PAIRADDR');
        if(address(factory) == address(0)){
            Return (0, 0);
        }
        uint256[8] memory dataAll;
        (dataAll[6], dataAll[7],) = IUniswapV2Pair(_pairAddr).getReserves();
        //require(dataAll[6] > 0, 'calcTokenOut: INSUFFICIENT_RESERVE0');
        //require(dataAll[7] > 0, 'calcTokenOut: INSUFFICIENT_RESERVE1');
        if(dataAll[6] == 0){
            Return (0, 0);
        }
        if(dataAll[7] == 0){
            Return (0, 0);
        }
        address[2] memory allPairAddr;
        allPairAddr[0] = factory.getPair(_tokenAddr,IUniswapV2Pair(_pairAddr).token0());
        //require(allPairAddr[0] != address(0), 'calcToken: INVALID_PAIR0');
        if(allPairAddr[0] == address(0)){
            Return (0, 0);
        }
        dataAll[0] = getPrice(allPairAddr[0],_tokenAddr);
        allPairAddr[1] = factory.getPair(_tokenAddr,IUniswapV2Pair(_pairAddr).token1());
        //require(allPairAddr[1] != address(0), 'calcToken: INVALID_PAIR1');
        if(allPairAddr[1] == address(0)){
            Return (0, 0);
        }
```

```
        dataAll[1] = getPrice(allPairAddr[1],_tokenAddr);

        dataAll[2] = _tokenAmount.mul(dataAll[1]).mul(dataAll[6]).div(dataAll[0].mul(dataAll[7]).add(dataAll[1].mul(dataAll[6])));

        dataAll[3] = _tokenAmount.sub(dataAll[2]);
        dataAll[4] = getAmountOut(allPairAddr[0],_tokenAddr,dataAll[2]);
        dataAll[5] = getAmountOut(allPairAddr[1],_tokenAddr,dataAll[3]);
        return (dataAll[4],dataAll[5]);
    }

    function calcTokenXOut(address _pairAddr,address _tokenAddr,uint256 _tokenAmount,uint256 tokenType) public view returns (uint256,uint256) {
        IUniswapV2Factory factory = IUniswapV2Factory(IUniswapV2Pair(_pairAddr).factory());
        //require(address(factory) != address(0), 'calcTokenXOut: INSUFFICIENT_PAIRADDR');
        if(address(factory) == address(0)){
            Return (0, 0);
        }
        uint256[5] memory dataAll;
        (dataAll[0], dataAll[1],) = IUniswapV2Pair(_pairAddr).getReserves();
        //require(dataAll[0] > 0, 'calcTokenXOut: INSUFFICIENT_RESERVE0');
        //require(dataAll[1] > 0, 'calcTokenXOut: INSUFFICIENT_RESERVE1');
        if(dataAll[0] == 0){
            Return (0, 0);
        }
        if(dataAll[1] == 0){
            Return (0, 0);
        }
        // (reserv_USDT * amount / (reserv_USDT + reserv_TEN) )
        dataAll[2] = _tokenAmount.div(2);
```

```
            dataAll[3] = _tokenAmount.sub(dataAll[2]);
            dataAll[4] = getAmountOut(_pairAddr,_tokenAddr,dataAll[3]);
            if(tokenType == 0){
                return (dataAll[2],dataAll[4]);
            }else{
                return (dataAll[4],dataAll[2]);
            }
        }
        .
}
// TenetProxyInner.sol
contract TenetProxyInner is Ownable {
        using SafeMath for uint256;
        using SafeERC20 for IERC20;

        uint256 public constant MINLPTOKEN_AMOUNT = 10;
        uint256 public constant MINWEALTH_AMOUNT = 1000000000000000000;
        Tenet public tenet;
        TenetMine public tenetmine;
        address public wethAddr;
        address public wusdtAddr;
        IUniswapV2Factory public uniFactory;
        constructor(Tenet _tenet,address _weth,address _wusdt) public {
            tenet = _tenet;
            tenetmine = tenet.tenMineCalc();
            wethAddr = _weth;
            wusdtAddr = _wusdt;
            uniFactory                                                    =
IUniswapV2Factory(IUniswapV2Pair(address(tenet.lpTokenTen())).factory());
        }
        function set_tenet(Tenet _tenet) public onlyOwner {
            tenet = _tenet;
            tenetmine = tenet.tenMineCalc();
```

```
    }

    Function getTenPoolNewInfo() public view Returns (UINT256 [6] Memory RetDatas1, UINT256 [6] Memory RetDatas2, UINT256 [8] Memory RetDatas3) {// knownSec returns () public view Returns (UINT256 [6] Memory RetDatas2, UINT256 [8] Memory RetDatas3)
        retDatas1 = getTenUserPool(); //(lastRewardBlock,accTenPerShare,allocPoint,lpTokenTotalAmount,totalAllocPoint,newBlockTen);
        retDatas2 = getTenProjectPool(); //(lastRewardBlock,accTenPerShare,allocPoint,lpTokenTotalAmount,totalAllocPoint,newBlockTen);
        retDatas3 = getPoolPriceInfo(address(tenet.lpTokenTen()),address(tenet.ten())); //(lpSupply,reserve0,reserve1,pricetype,price0,price1,tokeneth,tokenusdt);
    }

    Function getTokenPoolNewInfo(uint256_pid) public view returns (uint256[8] Memory RetDatas1,uint256[8] Memory RetDatas3) {// knownSec // getTokenPoolNewInfo(uint256_pid) public view returns (uint256[8] Memory RetDatas3,uint256[8] Memory RetDatas3) {// knownSec //
        retDatas1 = getPoolInfo(_pid); //(lastRewardBlock,lpTokenTotalAmount,accTokenPerShare,accTenPerShare,userCount,tenLPTokenAmount,rewardTenDebt,mineTokenAmount)
        newTenPerBlock = getTenPerBlockByProjectID(_pid);
        (address pairAddr,address tokenAddr, , , , , , ) = tenet.poolSettingInfo(_pid);
        retDatas3 = getPoolPriceInfo(pairAddr,tokenAddr); //(lpSupply,reserve0,reserve1,pricetype,price0,price1,tokeneth,tokenusdt);
    }

    function getTenPoolBasicInfo() public view returns (address[5] memory retData1,uint256[3] memory retData2,uint256[8] memory retData3,uint256[50] memory retData4,string memory retData5,string memory retData6,string memory retData7) {
        address pairAddr = address(tenet.lpTokenTen());
        address tokenAddr = address(tenet.ten());
```

```
        (retData1,retData2,retData5,retData6,retData7) =
getPairBasicInfo(pairAddr,tokenAddr);
        (retData3,retData4) = getTenPoolMineInfo();
    }

    function getTokenPoolBasicInfo(uint256 _pid) public view returns (address[5] memory
retData1,uint256[3] memory retData2,address[3] memory retData3,uint256[6] memory
retData4,string memory retData5,string memory retData6,string memory retData7) {
        (address pairAddr,address tokenAddr, , , , , , ) = tenet.poolSettingInfo(_pid);
        (retData1,retData2,retData5,retData6,retData7) =
getPairBasicInfo(pairAddr,tokenAddr);
        (retData3,retData4) = getTokenPoolMineInfo(_pid);
    }

    Function getPoolPriceInfo(Address PairAddr, Address TokenAddr) public view returns
(UINT256 [8] Memory Retdatas) {// knownSec //
        address factory = IUniswapV2Pair(pairAddr).factory();
        address token0Addr = IUniswapV2Pair(pairAddr).token0();
        address token1Addr = IUniswapV2Pair(pairAddr).token1();
        retDatas[0] = IUniswapV2Pair(pairAddr).totalSupply();
        (retDatas[1], retDatas[2],) = IUniswapV2Pair(pairAddr).getReserves();
        (retDatas[3],retDatas[4],retDatas[5]) =
calcTokenPrice(IUniswapV2Factory(factory),token0Addr,token1Addr);
        (retDatas[6],retDatas[7]) = calcPrice(uniFactory,tokenAddr);
    }
    function getPairBasicInfo(address pairAddr,address tokenAddr) public view returns
(address[5] memory retData1,uint256[3] memory retData2,string memory retData3,string
memory retData4,string memory RetData5) {// knownSec // Get basic information about trades
        retData1[0] = IUniswapV2Pair(pairAddr).factory();
        retData1[1] = pairAddr;
        retData1[2] = tokenAddr;
        retData1[3] = IUniswapV2Pair(pairAddr).token0();
        retData1[4] = IUniswapV2Pair(pairAddr).token1();
```

```
        (retData2[0],retData3) = getTokenInfo(retData1[2]);
        (retData2[1],retData4) = getTokenInfo(retData1[3]);
        (retData2[2],retData5) = getTokenInfo(retData1[4]);
    }

    function getTenUserPool() public view returns (uint256[6] memory) {
        uint256[6] memory retDatas;
        (retDatas[0],retDatas[1],retDatas[2],retDatas[3]) = tenet.tenUserPool();
        retDatas[4] = tenet.totalAllocPoint();
        retDatas[5] = getTenPerBlockByUser();
        return retDatas;
//(lastRewardBlock,accTenPerShare,allocPoint,lpTokenTotalAmount,totalAllocPoint,newBlockTen)
;
    }

    function getTenProjectPool() public view returns (uint256[6] memory) {
        uint256[6] memory retDatas;
        (retDatas[0],retDatas[1],retDatas[2],retDatas[3]) = tenet.tenProjectPool();
        retDatas[4] = tenet.totalAllocPoint();
        retDatas[5] = getTenPerBlockByProject();
        return retDatas;
//(lastRewardBlock,accTenPerShare,allocPoint,lpTokenTotalAmount,totalAllocPoint,newBlockTen)
;
    }

    Function getTenPoolMineInfo() public view returns (UINT256 [8] Memory Retdata1,
UINT256 [50] Memory Retdata2) {// knownSec returns () public view returns (UINT256 [8]
Memory Retdata2)
        retData1[0] = tenetmine.startBlock();
        retData1[1] = tenetmine.endBlock();
        retData1[2] = tenetmine.bonusEndBlock();
        retData1[3] = tenetmine.bonus_multiplier();
        retData1[4] = tenetmine.bonusTenPerBlock();
```

```
            retData1[5] = tenetmine.subBlockNumerPeriod();
            retData1[6] = tenetmine.totalSupply();
            retData1[7] = tenetmine.getMinePeriodCount();
            for(uint256 i=0; i<tenetmine.getMinePeriodCount(); i++){
                if(i >= 50){
                    break;
                }
                (retData2[i], )= tenetmine.allMinePeriodInfo(i);
            }
        }

        Function getTokenPoolmineInfo (UINT256_PID) public view returns (address[3] memory retData1, UINT256 [6] memory retData2) {// knownSec //
            (retData1) = getPoolSettingInfo1(_pid);
            (retData2) = getPoolSettingInfo2(_pid);
        }
        function getPoolSettingInfo1(uint256 _pid) public view returns (address[3] memory retData1) {
            (retData1[0],retData1[1],retData1[2],,,,,,) = tenet.poolSettingInfo(_pid);
        }
        function getPoolSettingInfo2(uint256 _pid) public view returns (uint256[6] memory retData2) {
            (,,,retData2[0],retData2[1],retData2[2],retData2[3],retData2[4],retData2[5]) = tenet.poolSettingInfo(_pid);
        }
        function getPoolInfo(uint256 _pid) public view returns (uint256[8] memory retData3) {

(retData3[0],retData3[1],retData3[2],retData3[3],retData3[4],retData3[5],retData3[6],retData3[7]) = tenet.poolInfo(_pid);
        }
        function getTokenInfo(address tokenAddr) public view returns (uint256 retData1,string memory retData2) {
            retData1 = ERC20(tokenAddr).decimals();
```

```
        retData2 = ERC20(tokenAddr).symbol();
    }

    Function calcPrice(IUNISWAPV2Factory _Factory,address tokenAddr) public view returns (uint256,uint256) {// knownSec //
        uint256 price0 = calcTokenWealth(_factory,tokenAddr,wethAddr);
        uint256 price1 = calcTokenWealth(_factory,tokenAddr,wusdtAddr);
        return (price0,price1);
    }

    Function calcTokenPrice(IUNISWAPV2Factory _Factory,address token0Addr,address token1Addr) public view returns (uint256,uint256,uint256)
        uint256 pricetype = 0;
        uint256 price0 = 0;
        uint256 price1 = 0;
        price0 = calcTokenWealth(_factory,token0Addr,wethAddr);
        if(price0 == 0){
            price1 = calcTokenWealth(_factory,token1Addr,wethAddr);
            if(price1 == 0){
                pricetype = 1;
                price0 = calcTokenWealth(_factory,token0Addr,wusdtAddr);
                if(price0 == 0){
                    price1 = calcTokenWealth(_factory,token1Addr,wusdtAddr);
                }
            }
        }
        return (pricetype,price0,price1);
    }

    function calcETHPrice() public view returns (uint256) {
        IUniswapV2Pair pair = IUniswapV2Pair(uniFactory.getPair(wethAddr,wusdtAddr));
        if (address(pair) == address(0)) {
            return 0;
```

```
        }
        address token0 = pair.token0();
        (uint reserve0, uint reserve1,) = pair.getReserves();
        if(token0 == wethAddr){
            return reserve1.mul(MINWEALTH_AMOUNT).div(reserve0);
        }
        return reserve0.mul(MINWEALTH_AMOUNT).div(reserve1);
    }

    function calcTokenWealth(IUniswapV2Factory _factory,address token,address wealth)
public view returns (uint256) {
        if (token == wealth) {
            return MINWEALTH_AMOUNT;
        }
        IUniswapV2Pair pair = IUniswapV2Pair(_factory.getPair(token, wealth));
        if (address(pair) == address(0)) {
            return 0;
        }
        (uint reserve0, uint reserve1,) = pair.getReserves();
        if(token == pair.token0()){
            return reserve1.mul(MINWEALTH_AMOUNT).div(reserve0);
        }
        return reserve0.mul(MINWEALTH_AMOUNT).div(reserve1);
    }

    function getTenPerBlockByUser() public view returns (uint256 tenReward) {
        uint256[2] memory allTmpData;     //allocPoint,lpSupply
        (,,allTmpData[0],allTmpData[1]) = tenet.tenUserPool();
        if (allTmpData[1] < MINLPTOKEN_AMOUNT) {
            return 0;
        }
        tenReward = tenetmine.calcMineTenReward(block.number-1, block.number);
        tenReward = tenReward.mul(allTmpData[0]).div(tenet.totalAllocPoint());
```

```
    }

    function getTenPerBlockByProject() public view returns (uint256 tenReward) {
        uint256[3] memory allTmpData;     //lpTokenAmount,allocPoint,tenLPTokenAmount
        ( , ,allTmpData[0],allTmpData[1]) = tenet.tenProjectPool();
        if (allTmpData[1] < MINLPTOKEN_AMOUNT) {
            return 0;
        }
        tenReward = tenetmine.calcMineTenReward(block.number-1, block.number);
        tenReward = tenReward.mul(allTmpData[0]).div(tenet.totalAllocPoint());
    }

    function getTenPerBlockByProjectID(uint _pid) public view returns (uint256 tenReward) {
        uint256[4] memory allTmpData;     //lpTokenAmount,allocPoint,tenLPTokenAmount
        ( ,allTmpData[0], , , ,allTmpData[3], , ) = tenet.poolInfo(_pid);
        if(allTmpData[0] < MINLPTOKEN_AMOUNT){
            return 0;
        }
        if (allTmpData[3] < MINLPTOKEN_AMOUNT) {
            return 0;
        }
        ( , ,allTmpData[1],allTmpData[2]) = tenet.tenProjectPool();
        tenReward = tenetmine.calcMineTenReward(block.number-1, block.number);
        tenReward                                                              =
tenReward.mul(allTmpData[3]).mul(allTmpData[1]).div(tenet.totalAllocPoint()).div(allTmpData[
2]);
    }
}
```

Safety advice: none.

3.5. Token conversion function [through]

Audit analysis: The token conversion function is implemented in the tenetswap.sol contract, which is mainly used to add liquidity, query the price of tokens, the amount of conversion, and convert the transaction tokens, etc.

```
contract TenetSwap is Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    Tenet public tenetAddr;
    address public wethAddr;

    modifier ensure(uint deadline) {
        require(deadline >= block.timestamp, 'TenetSwap: EXPIRED');
        _;
    }
    event TransferTokenToLPToken(address indexed user, uint256 indexed pid, address tokenAddr,uint256 tokenAmount,address lpTokenAddr, uint256 lpTenAmount);
    event TransferTokensToLPToken(address indexed user, uint256 indexed pid, uint256 token0Amount,uint256 token1Amount,address lpTokenAddr, uint256 lpTenAmount);
    event ChangeLPToken(address indexed user,address lpTokenAddr,uint256 token0Amount,uint256 token1Amount,uint256 lpTenAmount);

    constructor(address _tenetAddr,address _wethAddr) public {
        tenetAddr = Tenet(_tenetAddr);
        wethAddr = _wethAddr;
    }
    Function set_tenet(Tenet _tenet) public onlyOwner {// knownSec
        tenetAddr = _tenet;
    }
```

```
    receive() external payable {
        assert(msg.sender == wethAddr);  // only accept ETH via fallback from the WETH contract
    }
    function _calcLiquidAmountIn(address _pairAddr,uint256[2] memory _amountDesired,uint256 _amountMinRate) internal virtual view returns (uint amount0, uint amount1) {
        require(_amountMinRate <= 1000, 'addLiquidity: INSUFFICIENT_AMOUNT_MINRATE');
        uint256[2] memory _amountMin;
        _amountMin[0] = _amountDesired[0].mul(_amountMinRate).div(1000);
        _amountMin[1] = _amountDesired[1].mul(_amountMinRate).div(1000);
        uint256[2] memory _reserve;
        (_reserve[0],_reserve[1],) = IUniswapV2Pair(_pairAddr).getReserves();
        uint amount1Optimal = _amountDesired[0].mul(_reserve[1]).div(_reserve[0]);
        if (amount1Optimal <= _amountDesired[1]) {
            require(amount1Optimal >= _amountMin[1], '_addLiquidity: INSUFFICIENT_1_AMOUNT');
            (amount0, amount1) = (_amountDesired[0], amount1Optimal);
        } else {
            uint amount0Optimal = _amountDesired[1].mul(_reserve[0]).div(_reserve[1]);
            assert(amount0Optimal <= _amountDesired[0]);
            require(amount0Optimal >= _amountMin[0], '_addLiquidity: INSUFFICIENT_0_AMOUNT');
            (amount0, amount1) = (amount0Optimal, _amountDesired[1]);
        }
    }
    function _transferToPair(address _tokenAddr,address _fromAddr, address _toAddr, uint256 _value) internal {
        if(_tokenAddr == wethAddr){
            IWETH(_tokenAddr).transfer(_toAddr, _value);
        }else{
            if(_fromAddr == address(this)){
```

```
                IERC20(_tokenAddr).transfer(_toAddr, _value);
            }else{
                IERC20(_tokenAddr).transferFrom(_fromAddr,_toAddr, _value);
            }
        }
    }
    function _returnToUser(address _tokenAddr,address _fromAddr,uint256 _value) internal{
        if(_value > 0){
            if(_tokenAddr == wethAddr){
                IWETH(_tokenAddr).withdraw(_value);
                msg.sender.transfer(_value);
            }else{
                if(_fromAddr == address(this)){
                    IERC20(_tokenAddr).transfer(msg.sender, _value);
                }
            }
        }
    }
    Function _addLiquidity(address _fromAddr,address _pairAddr,uint256[2] memory
_amountDesired,address to,uint256 _amountMinRate) internal returns (uint256) {// knownSec //
        address[2] memory _tokenAddr;
        _tokenAddr[0] = IUniswapV2Pair(_pairAddr).token0();
        _tokenAddr[1] = IUniswapV2Pair(_pairAddr).token1();
        (uint256 amountA,uint256 amountB) = _calcLiquidAmountIn(_pairAddr,
_amountDesired, _amountMinRate);
        _transferToPair(_tokenAddr[0],_fromAddr,_pairAddr,amountA);
        _transferToPair(_tokenAddr[1],_fromAddr,_pairAddr,amountB);
        uint256 liquidity = IUniswapV2Pair(_pairAddr).mint(to);
        _returnToUser(_tokenAddr[0],_fromAddr,_amountDesired[0].sub(amountA));
        _returnToUser(_tokenAddr[1],_fromAddr,_amountDesired[1].sub(amountB));
        return liquidity;
    }
    Function getPrice(address _pairAddr, address _fromAddr) public view returns (uint256) {//
```

```
knownSec //
            (uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(_pairAddr).getReserves();
            if(_fromAddr == IUniswapV2Pair(_pairAddr).token0()){
                return reserve1.mul(1e12).div(reserve0);
            }else{
                return reserve0.mul(1e12).div(reserve1);
            }
        }
        Function getAmountOut(Address _PairAddr, Address _FromAddr, Uint Amountin) public
view virtual returns (UINT256)
            require(amountIn > 0, 'getAmountOut: INSUFFICIENT_INPUT_AMOUNT');
            (uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(_pairAddr).getReserves();
            require(reserve0 > 0 && reserve1 > 0, 'getAmountOut:
INSUFFICIENT_LIQUIDITY');
            if(_fromAddr == IUniswapV2Pair(_pairAddr).token0()){
                uint amountInWithFee = amountIn.mul(997);
                uint numerator = amountInWithFee.mul(reserve1);
                uint denominator = reserve0.mul(1000).add(amountInWithFee);
                return numerator.div(denominator);
            }else{
                uint amountInWithFee = amountIn.mul(997);
                uint numerator = amountInWithFee.mul(reserve0);
                uint denominator = reserve1.mul(1000).add(amountInWithFee);
                return numerator.div(denominator);
            }
        }
        Function _swapToken(address _pairAddr, address _fromAddr,uint256 _tokenAmount)
internal returns (uint256) {//knownsec//
            uint256 tokenAmountOut = getAmountOut(_pairAddr,_fromAddr,_tokenAmount);
            if(_fromAddr == wethAddr){
                IWETH(_fromAddr).transfer(_pairAddr, _tokenAmount);
            }else{
                IERC20(_fromAddr).transfer(_pairAddr, _tokenAmount);
```

```
            }
            if(_fromAddr == IUniswapV2Pair(_pairAddr).token0()){
                IUniswapV2Pair(_pairAddr).swap(0, tokenAmountOut, address(this), new bytes(0));
            }else{
                IUniswapV2Pair(_pairAddr).swap(tokenAmountOut, 0, address(this), new bytes(0));
            }
            return tokenAmountOut;
        }
        function _transferTokensOut(address _pairAddr,address _tokenAddr,uint256 _tokenAmount) internal returns (uint256,uint256) {
            IUniswapV2Factory factory = IUniswapV2Factory(IUniswapV2Pair(_pairAddr).factory());
            require(address(factory) != address(0), 'transferTokensOut: INSUFFICIENT_PAIR');
            uint256[8] memory dataAll;
            (dataAll[6], dataAll[7],) = IUniswapV2Pair(_pairAddr).getReserves();
            require(dataAll[6] > 0, 'transferTokensOut: INSUFFICIENT_RESERVE0');
            require(dataAll[7] > 0, 'transferTokensOut: INSUFFICIENT_RESERVE1');
            address[2] memory allPairAddr;
            allPairAddr[0] = factory.getPair(_tokenAddr,IUniswapV2Pair(_pairAddr).token0());
            require(allPairAddr[0] != address(0), 'transferTokensOut: INVALID_PAIR0');
            dataAll[0] = getPrice(allPairAddr[0],_tokenAddr);
            allPairAddr[1] = factory.getPair(_tokenAddr,IUniswapV2Pair(_pairAddr).token1());
            require(allPairAddr[1] != address(0), 'transferTokensOut: INVALID_PAIR1');
            dataAll[1] = getPrice(allPairAddr[1],_tokenAddr);
            dataAll[2] = _tokenAmount.mul(dataAll[1]).mul(dataAll[6]).div(dataAll[0].mul(dataAll[7]).add(dataAll[1].mul(dataAll[6])));
            dataAll[3] = _tokenAmount.sub(dataAll[2]);
            dataAll[4] = _swapToken(allPairAddr[0],_tokenAddr,dataAll[2]);
            dataAll[5] = _swapToken(allPairAddr[1],_tokenAddr,dataAll[3]);
            return (dataAll[4],dataAll[5]);
```

```
    }
    function _transferTokenXOut(address _pairAddr,address _tokenAddr,uint256 _tokenAmount,uint256 tokenType) internal returns (uint256,uint256) {
        IUniswapV2Factory factory = IUniswapV2Factory(IUniswapV2Pair(_pairAddr).factory());
        require(address(factory) ! = address(0), 'transferTokenXOut: INSUFFICIENT_PAIR');
        uint256[4] memory dataAll;
        (dataAll[0], dataAll[1],) = IUniswapV2Pair(_pairAddr).getReserves();
        require(dataAll[0] > 0, 'transferTokenXOut: INSUFFICIENT_RESERVE0');
        require(dataAll[1] > 0, 'transferTokenXOut: INSUFFICIENT_RESERVE1');
        dataAll[2] = _tokenAmount.div(2);
        dataAll[3] = _swapToken(_pairAddr,_tokenAddr,dataAll[2]);
        if(tokenType == 0){
            return (dataAll[2],dataAll[3]);
        }else{
            return (dataAll[3],dataAll[2]);
        }
    }
    function _transferLPToken(uint256 _poolType,uint256 _pid,address _pairAddr,uint256[2] memory _tokenAmountOut,uint256 _amountMinRate) internal returns (uint256) {
        uint liquidity = _addLiquidity(address(this),_pairAddr,_tokenAmountOut,address(this),_amountMinRate);
        IERC20(_pairAddr).approve(address(tenetAddr),liquidity);
        if(_poolType == 0){
            tenetAddr.depositTenByUserFrom(msg.sender,liquidity);
        }else{
            tenetAddr.depositLPTokenFrom(msg.sender,_pid,liquidity);
        }
        return liquidity;
    }
    function transferTokenToLPToken(uint256 _poolType,uint256 _pid,address _pairAddr,address _tokenAddr,uint256 _tokenAmount,uint256 _amountMinRate,uint256 deadline) public virtual ensure(deadline) {
```

```
        if(_tokenAddr ! = wethAddr){
            IERC20(_tokenAddr).transferFrom(msg.sender, address(this),_tokenAmount);
        }
        uint256[2] memory tokenAmountOut;
        if(_tokenAddr == IUniswapV2Pair(_pairAddr).token0()){
            (tokenAmountOut[0],tokenAmountOut[1]) =
_transferTokenXOut(_pairAddr,_tokenAddr,_tokenAmount,0);
        }else if(_tokenAddr == IUniswapV2Pair(_pairAddr).token1()){
            (tokenAmountOut[0],tokenAmountOut[1]) =
_transferTokenXOut(_pairAddr,_tokenAddr,_tokenAmount,1);
        }else{
            (tokenAmountOut[0],tokenAmountOut[1]) =
_transferTokensOut(_pairAddr,_tokenAddr,_tokenAmount);
        }
        uint liquidity =
_transferLPToken(_poolType,_pid,_pairAddr,tokenAmountOut,_amountMinRate);
        emit
TransferTokenToLPToken(msg.sender,_pid,_tokenAddr,_tokenAmount,_pairAddr,liquidity);
    }
    function transferETHToLPToken(uint256 _poolType,uint256 _pid,address
_pairAddr,uint256 _amountMinRate,uint256 deadline) external virtual payable ensure(deadline) {
        IWETH(wethAddr).deposit{value: msg.value}();

transferTokenToLPToken(_poolType,_pid,_pairAddr,wethAddr,msg.value,_amountMinRate,deadli
ne);
    }
    function transferTokensToLPToken(uint256 _poolType,uint256 _pid,address
_pairAddr,uint256 _token0Amount,uint256 _token1Amount,uint256 _amountMinRate,uint256
deadline) public virtual ensure(deadline) {
        uint256[2] memory tokenAmountOut;
        tokenAmountOut[0] = _token0Amount;
        tokenAmountOut[1] = _token1Amount;
        if(wethAddr ! = IUniswapV2Pair(_pairAddr).token0()){
```

```
            IERC20(IUniswapV2Pair(_pairAddr).token0()).transferFrom(msg.sender,
address(this), tokenAmountOut[0]);
        }
        if(wethAddr ! = IUniswapV2Pair(_pairAddr).token1()){
            IERC20(IUniswapV2Pair(_pairAddr).token1()).transferFrom(msg.sender,
address(this), tokenAmountOut[1]);
        }
        uint                                liquidity                                =
_transferLPToken(_poolType,_pid,_pairAddr,tokenAmountOut,_amountMinRate);
        emit
TransferTokensToLPToken(msg.sender,_pid,tokenAmountOut[0],tokenAmountOut[1],_pairAddr,li
quidity);
    }
    function transferETHsToLPToken(uint256 _poolType,uint256 _pid,address
_pairAddr,uint256 _tokenAmount,uint256 _amountMinRate,uint256 deadline) external virtual
payable ensure(deadline) {
        IWETH(wethAddr).deposit{value: msg.value}();
        if(wethAddr == IUniswapV2Pair(_pairAddr).token0()){

transferTokensToLPToken(_poolType,_pid,_pairAddr,msg.value,_tokenAmount,_amountMinRate,d
eadline);
        }else{

transferTokensToLPToken(_poolType,_pid,_pairAddr,_tokenAmount,msg.value,_amountMinRate,d
eadline);
        }
    }
    function changeLPToken(address _pairAddr,uint256[2] memory _tokenAmountOut,uint256
_amountMinRate,uint256 deadline) public ensure(deadline){
        uint                                liquidity                                =
_addLiquidity(msg.sender,_pairAddr,_tokenAmountOut,msg.sender,_amountMinRate);
        emit
ChangeLPToken(msg.sender,_pairAddr,_tokenAmountOut[0],_tokenAmountOut[1],liquidity);
```

```
    }
    function changeWethLPToken(address _pairAddr,uint256 _tokenAmount,uint256 _amountMinRate,uint256 deadline) external payable ensure(deadline){
        uint256[2] memory tokenAmountOut;
        if(wethAddr == IUniswapV2Pair(_pairAddr).token0()){
            tokenAmountOut[0] = msg.value;
            tokenAmountOut[1] = _tokenAmount;
        }else{
            tokenAmountOut[0] = _tokenAmount;
            tokenAmountOut[1] = msg.value;
        }
        IWETH(wethAddr).deposit{value: msg.value}();
        changeLPToken(_pairAddr,tokenAmountOut,_amountMinRate,deadline);
    }
}
```

Safety advice: none.

4. Basic code vulnerability detection

4.1. Compiler version security [pass]

Check that a secure compiler version is used in the contract code implementation

Test results: After testing, the smart contract code specified the compiler version 0.6.12 or above, there is no such security problem.

Safety advice: none.

4.2. Redundant code [pass]

Check if the contract code implementation contains redundant code

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.3. The use of secure arithmetic libraries [pass]

Check that the contract code implementation uses the SAFEMATH security arithmetic library

Test results: After testing, the smart contract code has used SAFEMATH security arithmetic library, and there is no such security problem.

Safety advice: none.

4.4. Not Recommended Coding Method [Pass]

Check if there are any officially deprecated or deprecated encoding options in the contract code implementation

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.5. Fair use of require/assert

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

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.6. Fallback function security [pass]

Verify that the fallback function is used correctly in the contract code implementation

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.7. TX.Origin Authentication [Pass]

TX.origin is a global variable in Solidity that iterates through the call stack and returns the address of the account from which the call (or transaction) was originally sent. Using this variable for authentication in a smart contract makes the contract vulnerable to phishing attacks like this.

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.8. Owner permission control

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

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.9. Gas consumption detection [pass]

Check if the gas consumption exceeds the block maximum limit

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.10. Call injection attack [pass]

When a call function is called, strict permission control should be done, or the call function should be written to death.

Test results: After testing, the smart contract does not use the call function, so there is no such vulnerability.

Safety advice: none.

4.11. Low level function security [pass]

Check for 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 current calling contract

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.12. Added token vulnerability [pass]

Check if there are functions in the token contract that could increase the total amount of tokens after initializing the total amount of tokens.

Test result: After testing, the smart contract code has the function of issuing additional tokens, but it passed because liquidity mining requires issuing additional tokens.

Safety advice: none.

4.13. Access control defect detection [pass]

Reasonable permissions should be set for different functions in the contract

Check whether all functions in the contract correctly use keywords such as public and private for visibility modification, and check whether the contract correctly defines and uses modifier to restrict access to key functions to avoid problems caused by overstepping authority.

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.14. Numerical overflow detection [pass]

The arithmetic problem in smart contract refers to integer overflow and integer underoverflow.

Solidity can handle up to 256 bits of numbers (2^256-1). Increasing the maximum number by 1 will overflow to 0.Similarly, when the number is of unsigned type, 0 minus 1 will overflow down to the maximum numeric value.

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

Test results: After testing, the smart contract code does not have this

security problem.

Safety advice: none.

4.15. 8. An error of arithmetical precision.

Solidity as a programming language and common programming language similar data structure design, such as: variables, constants, and functions, arrays, functions, structure and so on, Solidity and common programming language also has a larger difference - no floating-point Solidity, Solidity and all the numerical computing results can only be an integer, decimal will not happen, also not allowed to define the decimal data type.Numerical operation in the contract is essential, and the design of numerical operation may cause relative errors. For example, the parallel operation: 5/2*10=20, and 5*10/2=25, thus generating errors. When the data is larger, the errors will be larger and more obvious.

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.16. Incorrect use of random numbers

In intelligent contracts may need to use a random number, although the Solidity of functions and variables can access the value of the unpredictable obviously such as block. The number and block. The timestamp, but they usually or more open than it looks, or is affected by the miners, that is, to some extent,

these random Numbers is predictable, so a malicious user can copy it and usually rely on its unpredictability to attack the function.

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.17. Unsecure interface usage [pass]

Check if an insecure interface is used in the contract code implementation

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.18. Variable override [pass]

Check for security issues caused by variable overwriting in the contract code implementation

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.19. Uninitialized store pointer [pass]

Solidity allows a specific data structure to be struct struct and local variables within functions to be stored in Storage or Memory by default.

Solidity allows a pointer to point to an uninitialized reference. An uninitialized local stroage will cause a variable to point to another stored variable, resulting in variable overwrite, and even more serious consequences. In development you should avoid initializing struct variables in functions.

Test results: After testing, the smart contract code does not have this problem.

Safety advice: none.

4.20. Return value call validation [pass]

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

In Solidity, there are transfer(), send(), call.value() and other transfer methods, which can all be used to send Ether to a certain address. The difference lies in: when the transfer fails to send, throw it and roll back the state;Only 2300gas will be passed to prevent reentrant attacks. Send returns false if it fails; Only 2300gas will be passed to prevent reentrant attacks. Call. Value returns false when it fails to send; Passing all available GAS calls (which can be restricted by passing the GAS_VALUE parameter) does not effectively prevent reentrant 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, possibly with unexpected results due to the Ether send failure.

Test results: After testing, the smart contract code does not have this

security problem.

Safety advice: none.

4.21. Transaction order depends on [pass]

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

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none

4.22. Timestamp Dependency Attacks [Pass]

The timestamp of the data block is usually the local time of the miner, which can vary by about 900 seconds. When another node accepts a new block, it only needs to verify that the timestamp is later than the previous block and within 900 seconds of the local time.A miner can profit by setting the timestamp of a block to meet conditions as favorable to him as possible.

Check if there are any key functions in the contract code implementation that depend on timestamps

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.23. Denial of service attack [pass]

In the world of Ethereum, denial of service is deadly, and a smart contract subject to this type of attack may never return to its normal working state. A smart contract denial of service can be caused by a variety of reasons, including malicious behavior while being the recipient of a transaction, running out of gas by artificially increasing the gas required for computing functions, abusing access control to access private components of the smart contract, exploiting confusion and neglect, and so on.

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.24. False recharge vulnerability [pass]

The transfer function of the token contract checks the balance of the transfer promoter (msg.sender) with the if judging method. When balances[msg.sender] < value, it goes into the else logic part and returns false, and no exception is raised in the end. We believe that the gentle judging method of if/else only is a kind of unstrict coding method in the scene of transfer and other sensitive

functions.

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.25. Reentrant attack detection [pass]

The reentrant vulnerability is The most famous Ethereum smart contract vulnerability that led to The DAO hack.

The call.value() function in Solidity consumes all the gas it receives when it is used to send Ether, and there is a risk of a reentrant attack when calling the call.value() function to send Ether takes place before actually reducing the balance in the sender's account.

Test results: After testing, the smart contract code does not have this security problem.

Safety advice: none.

4.26. Replay Attack Detection [Pass]

If the contract involves the requirement of entrusted management, attention should be paid to the unreusability of verification to avoid replay attack

In the asset management system, there is often a situation of entrustment management. The principal gives the asset to the trustee for management, and the principal pays a certain fee to the trustee. This business scenario is also

common in smart contracts.

Test results: After testing, the smart contract does not use the call function, so there is no such vulnerability.

Safety advice: none.

4.27. Rearranged Attack Detection [Pass]

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

Test results: After testing, there is no relevant vulnerability in the smart contract code.

Safety advice: none.

5. Appendix A: Contract Code

Source of code for this test:

GovernorAlpha.sol

```
// SPDX-License-Identifier: MIT
// COPIED FROM https://github.com/compound-finance/compound-protocol/blob/master/contracts/Governance/GovernorAlpha.sol
// Copyright 2020 Compound Labs, Inc.
// Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
// 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
// 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
// 3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//
// Ctrl+f for XXX to see all the modifications.
// uint96s are changed to uint256s for simplicity and safety.

// XXX: pragma solidity ^0.5.16;
Pragma solidity 0.6.12;
pragma experimental ABIEncoderV2;

import "./TenetToken.sol";

contract GovernorAlpha {
    // @notice The name of this contract
    // XXX: string public constant name = "Compound Governor Alpha";
    string public constant name = "Tenet Governor Alpha";

    // @notice The number of votes in support of a proposal required in order for a quorum to be reached and for a vote to succeed
    // XXX: function quorumVotes() public pure returns (uint) { return 400000e18;      } // 400,000 = 4% of Comp
    function quorumVotes() public view returns (uint) { return govToken.totalSupply() / 25;     } // 4% of Supply

    // @notice The number of votes required in order for a voter to become a proposer
    // function proposalThreshold() public pure returns (uint) { return 100000e18;      } // 100,000 = 1% of Comp
    function proposalThreshold() public view returns (uint) { return govToken.totalSupply() / 100;      } // 1% of Supply

    // @notice The maximum number of actions that can be included in a proposal
    function proposalMaxOperations() public pure returns (uint) { return 10;     } // 10 actions

    // @notice The delay before voting on a proposal may take place, once proposed
    function votingDelay() public pure returns (uint) { return 1;     } // 1 block

    // @notice The duration of voting on a proposal, in blocks
    function votingPeriod() public pure returns (uint) { return 17280;      } // ~3 days in blocks (assuming 15s blocks)

    // @notice The address of the Compound Protocol Timelock
    TimelockInterface public timelock;

    // @notice The address of the Compound governance token
    // XXX: CompInterface public comp;
    TenetToken public govToken;

    // @notice The address of the Governor Guardian
    address public guardian;

    // @notice The total number of proposals
    uint public proposalCount;

    struct Proposal {
        // @notice Unique id for looking up a proposal
        uint id;

        // @notice Creator of the proposal
```

```
        address proposer;

        // @notice The timestamp that the proposal will be available for execution, set once the vote succeeds
        uint eta;

        // @notice the ordered list of target addresses for calls to be made
        address[] targets;

        // @notice The ordered list of values (i.e. msg.value) to be passed to the calls to be made
        uint[] values;

        // @notice The ordered list of function signatures to be called
        string[] signatures;

        // @notice The ordered list of calldata to be passed to each call
        bytes[] calldatas;

        // @notice The block at which voting begins: holders must delegate their votes prior to this block
        uint startBlock;

        // @notice The block at which voting ends: votes must be cast prior to this block
        uint endBlock;

        // @notice Current number of votes in favor of this proposal
        uint forVotes;

        // @notice Current number of votes in opposition to this proposal
        uint againstVotes;

        // @notice Flag marking whether the proposal has been canceled
        bool canceled;

        // @notice Flag marking whether the proposal has been executed
        bool executed;

        // @notice Receipts of ballots for the entire set of voters
        mapping (address => Receipt) receipts;
    }

    // @notice Ballot receipt record for a voter
    struct Receipt {
        // @notice Whether or not a vote has been cast
        bool hasVoted;

        // @notice Whether or not the voter supports the proposal
        bool support;

        // @notice The number of votes the voter had, which were cast
        uint256 votes;
    }

    // @notice Possible states that a proposal may be in
    enum ProposalState {
        Pending,
        Active,
        Canceled,
        Defeated,
        Succeeded,
        Queued,
        Expired,
        Executed
    }

    // @notice The official record of all proposals ever proposed
    mapping (uint => Proposal) public proposals;

    // @notice The latest proposal for each proposer
    mapping (address => uint) public latestProposalIds;

    // @notice The EIP-712 typehash for the contract's domain
    bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");

    // @notice The EIP-712 typehash for the ballot struct used by the contract
    bytes32 public constant BALLOT_TYPEHASH = keccak256("Ballot(uint256 proposalId,bool support)");

    // @notice An event emitted when a new proposal is created
    event ProposalCreated(uint id, address proposer, address[] targets, uint[] values, string[] signatures, bytes[] calldatas, uint startBlock, uint endBlock, string description);

    // @notice An event emitted when a vote has been cast on a proposal
    event VoteCast(address voter, uint proposalId, bool support, uint votes);

    // @notice An event emitted when a proposal has been canceled
    event ProposalCanceled(uint id);

    // @notice An event emitted when a proposal has been queued in the Timelock
```

```
    event ProposalQueued(uint id, uint eta);

    // @notice An event emitted when a proposal has been executed in the Timelock
    event ProposalExecuted(uint id);

    constructor(address timelock_, address govToken_, address guardian_) public {
        timelock = TimelockInterface(timelock_);
        govToken = TenetToken(govToken_);
        guardian = guardian_;
    }

    function propose(address[] memory targets, uint[] memory values, string[] memory signatures, bytes[] memory calldatas, string memory description) public returns (uint) {
        require(govToken.getPriorVotes(msg.sender, sub256(block.number, 1)) > proposalThreshold(), "GovernorAlpha::propose: proposer votes below proposal threshold");
        require(targets.length == values.length && targets.length == signatures.length && targets.length == calldatas.length, "GovernorAlpha::propose: proposal function information arity mismatch");
        require(targets.length != 0, "GovernorAlpha::propose: must provide actions");
        require(targets.length <= proposalMaxOperations(), "GovernorAlpha::propose: too many actions");

        uint latestProposalId = latestProposalIds[msg.sender];
        if (latestProposalId != 0) {
          ProposalState proposersLatestProposalState = state(latestProposalId);
          require(proposersLatestProposalState != ProposalState.Active, "GovernorAlpha::propose: one live proposal per proposer, found an already active proposal");
          require(proposersLatestProposalState != ProposalState.Pending, "GovernorAlpha::propose: one live proposal per proposer, found an already pending proposal");
        }

        uint startBlock = add256(block.number, votingDelay());
        uint endBlock = add256(startBlock, votingPeriod());

        proposalCount++;
        Proposal memory newProposal = Proposal({
            id: proposalCount,
            proposer: msg.sender,
            eta: 0,
            targets: targets,
            values: values,
            signatures: signatures,
            calldatas: calldatas,
            startBlock: startBlock,
            endBlock: endBlock,
            forVotes: 0,
            againstVotes: 0,
            canceled: false,
            executed: false
        });

        proposals[newProposal.id] = newProposal;
        latestProposalIds[newProposal.proposer] = newProposal.id;

        emit ProposalCreated(newProposal.id, msg.sender, targets, values, signatures, calldatas, startBlock, endBlock, description);
        return newProposal.id;
    }

    function queue(uint proposalId) public {
        require(state(proposalId) == ProposalState.Succeeded, "GovernorAlpha::queue: proposal can only be queued if it is succeeded");
        Proposal storage proposal = proposals[proposalId];
        uint eta = add256(block.timestamp, timelock.delay());
        for (uint i = 0; i < proposal.targets.length; i++) {
            _queueOrRevert(proposal.targets[i], proposal.values[i], proposal.signatures[i], proposal.calldatas[i], eta);
        }
        proposal.eta = eta;
        emit ProposalQueued(proposalId, eta);
    }

    function _queueOrRevert(address target, uint value, string memory signature, bytes memory data, uint eta) internal {
        require(!timelock.queuedTransactions(keccak256(abi.encode(target, value, signature, data, eta))), "GovernorAlpha::_queueOrRevert: proposal action already queued at eta");
        timelock.queueTransaction(target, value, signature, data, eta);
    }

    function execute(uint proposalId) public payable {
        require(state(proposalId) == ProposalState.Queued, "GovernorAlpha::execute: proposal can only be executed if it is queued");
        Proposal storage proposal = proposals[proposalId];
        proposal.executed = true;
        for (uint i = 0; i < proposal.targets.length; i++) {
            timelock.executeTransaction.value(proposal.values[i])(proposal.targets[i], proposal.values[i], proposal.signatures[i], proposal.calldatas[i], proposal.eta);
        }
        emit ProposalExecuted(proposalId);
```

```
    }

    function cancel(uint proposalId) public {
        ProposalState state = state(proposalId);
        require(state != ProposalState.Executed, "GovernorAlpha::cancel: cannot cancel executed proposal");

        Proposal storage proposal = proposals[proposalId];
        require(msg.sender == guardian || govToken.getPriorVotes(proposal.proposer, sub256(block.number, 1)) < proposalThreshold(), "GovernorAlpha::cancel: proposer above threshold");

        proposal.canceled = true;
        for (uint i = 0; i < proposal.targets.length; i++) {
            timelock.cancelTransaction(proposal.targets[i], proposal.values[i], proposal.signatures[i], proposal.calldatas[i], proposal.eta);
        }

        emit ProposalCanceled(proposalId);
    }

    function getActions(uint proposalId) public view returns (address[] memory targets, uint[] memory values, string[] memory signatures, bytes[] memory calldatas) {
        Proposal storage p = proposals[proposalId];
        return (p.targets, p.values, p.signatures, p.calldatas);
    }

    function getReceipt(uint proposalId, address voter) public view returns (Receipt memory) {
        return proposals[proposalId].receipts[voter];
    }

    function state(uint proposalId) public view returns (ProposalState) {
        require(proposalCount >= proposalId && proposalId > 0, "GovernorAlpha::state: invalid proposal id");
        Proposal storage proposal = proposals[proposalId];
        if (proposal.canceled) {
            return ProposalState.Canceled;
        } else if (block.number <= proposal.startBlock) {
            return ProposalState.Pending;
        } else if (block.number <= proposal.endBlock) {
            return ProposalState.Active;
        } else if (proposal.forVotes <= proposal.againstVotes || proposal.forVotes < quorumVotes()) {
            return ProposalState.Defeated;
        } else if (proposal.eta == 0) {
            return ProposalState.Succeeded;
        } else if (proposal.executed) {
            return ProposalState.Executed;
        } else if (block.timestamp >= add256(proposal.eta, timelock.GRACE_PERIOD())) {
            return ProposalState.Expired;
        } else {
            return ProposalState.Queued;
        }
    }

    function castVote(uint proposalId, bool support) public {
        return _castVote(msg.sender, proposalId, support);
    }

    function castVoteBySig(uint proposalId, bool support, uint8 v, bytes32 r, bytes32 s) public {
        bytes32 domainSeparator = keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(name)), getChainId(), address(this)));
        bytes32 structHash = keccak256(abi.encode(BALLOT_TYPEHASH, proposalId, support));
        bytes32 digest = keccak256(abi.encodePacked("\x19\x01", domainSeparator, structHash));
        address signatory = ecrecover(digest, v, r, s);
        require(signatory != address(0), "GovernorAlpha::castVoteBySig: invalid signature");
        return _castVote(signatory, proposalId, support);
    }

    function _castVote(address voter, uint proposalId, bool support) internal {
        require(state(proposalId) == ProposalState.Active, "GovernorAlpha::_castVote: voting is closed");
        Proposal storage proposal = proposals[proposalId];
        Receipt storage receipt = proposal.receipts[voter];
        require(receipt.hasVoted == false, "GovernorAlpha::_castVote: voter already voted");
        uint256 votes = govToken.getPriorVotes(voter, proposal.startBlock);

        if (support) {
            proposal.forVotes = add256(proposal.forVotes, votes);
        } else {
            proposal.againstVotes = add256(proposal.againstVotes, votes);
        }

        receipt.hasVoted = true;
        receipt.support = support;
        receipt.votes = votes;

        emit VoteCast(voter, proposalId, support, votes);
    }
```

```
        function __acceptAdmin() public {
            require(msg.sender == guardian, "GovernorAlpha::__acceptAdmin: sender must be gov guardian");
            timelock.acceptAdmin();
        }

        function __abdicate() public {
            require(msg.sender == guardian, "GovernorAlpha::__abdicate: sender must be gov guardian");
            guardian = address(0);
        }

        function __queueSetTimelockPendingAdmin(address newPendingAdmin, uint eta) public {
            require(msg.sender == guardian, "GovernorAlpha::__queueSetTimelockPendingAdmin: sender must be gov guardian");
            timelock.queueTransaction(address(timelock), 0, "setPendingAdmin(address)", abi.encode(newPendingAdmin), eta);
        }

        function __executeSetTimelockPendingAdmin(address newPendingAdmin, uint eta) public {
            require(msg.sender == guardian, "GovernorAlpha::__executeSetTimelockPendingAdmin: sender must be gov guardian");
            timelock.executeTransaction(address(timelock), 0, "setPendingAdmin(address)", abi.encode(newPendingAdmin), eta);
        }

        function add256(uint256 a, uint256 b) internal pure returns (uint) {
            uint c = a + b;
            require(c >= a, "addition overflow");
            return c;
        }

        function sub256(uint256 a, uint256 b) internal pure returns (uint) {
            require(b <= a, "subtraction underflow");
            return a - b;
        }

        function getChainId() internal pure returns (uint) {
            uint chainId;
            assembly { chainId := chainid() }
            return chainId;
        }
}

interface TimelockInterface {
        function delay() external view returns (uint);
        function GRACE_PERIOD() external view returns (uint);
        function acceptAdmin() external;
        function queuedTransactions(bytes32 hash) external view returns (bool);
        function queueTransaction(address target, uint value, string calldata signature, bytes calldata data, uint eta) external returns (bytes32);
        function cancelTransaction(address target, uint value, string calldata signature, bytes calldata data, uint eta) external;
        function executeTransaction(address target, uint value, string calldata signature, bytes calldata data, uint eta) external payable returns (bytes memory);
}
```

Migrations.sol

```
// SPDX-License-Identifier: MIT
Pragma solidity > = 0.4.25 < 0.7.0;

contract Migrations {
    address public owner;
    uint public last_completed_migration;

    modifier restricted() {
        if (msg.sender == owner) _;
    }

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

    function setCompleted(uint completed) public restricted {
        last_completed_migration = completed;
    }
}
```

MockERC20.sol

```
// SPDX-License-Identifier: MIT

Pragma solidity 0.6.12;


import "./openzeppelin/contracts/token/ERC20/ERC20.sol";
```

```
import "./openzeppelin/contracts/access/Ownable.sol";

contract MockERC20 is ERC20,Ownable {
    constructor(
        string memory name,
        string memory symbol,
        uint256 supply
    ) public ERC20(name, symbol) {
        _mint(msg.sender, supply);
    }
    function mint(address _to, uint256 _amount) public onlyOwner {
        _mint(_to, _amount);
    }
    function burn(address _account, uint256 _amount) public onlyOwner {
        _burn(_account, _amount);
    }
}
```

Tenet.sol

```
// SPDX-License-Identifier: MIT

Pragma solidity 0.6.12;


import "./openzeppelin/contracts/token/ERC20/IERC20.sol";
import "./openzeppelin/contracts/token/ERC20/SafeERC20.sol";
//import "./openzeppelin/contracts/utils/EnumerableSet.sol";
import "./openzeppelin/contracts/math/SafeMath.sol";
import "./openzeppelin/contracts/access/Ownable.sol";
import "./TenetToken.sol";
import "./TenetMine.sol";
// Tenet is the master of TEN. He can make TEN and he is a fair guy.
contract Tenet is Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    // Info of each user.
    struct UserInfo {
        uint256 amount;
        uint256 rewardTokenDebt;
        uint256 rewardTenDebt;
        uint256 lastBlockNumber;
        uint256 freezeBlocks;
        uint256 freezeTen;
    }
    // Info of each pool.
    struct PoolSettingInfo{
        address lpToken;
        address tokenAddr;
        address projectAddr;
        uint256 tokenAmount;
        uint256 startBlock;
        uint256 endBlock;
        uint256 tokenPerBlock;
        uint256 tokenBonusEndBlock;
        uint256 tokenBonusMultipler;
    }
    struct PoolInfo {
        uint256 lastRewardBlock;
        uint256 lpTokenTotalAmount;
        uint256 accTokenPerShare;
        uint256 accTenPerShare;
        uint256 userCount;
        uint256 amount;
        uint256 rewardTenDebt;
        uint256 mineTokenAmount;
    }

    struct TenPoolInfo {
        uint256 lastRewardBlock;
        uint256 accTenPerShare;
        uint256 allocPoint;
        uint256 lpTokenTotalAmount;
    }

    TenetToken public ten;
    TenetMine public tenMineCalc;
    IERC20 public lpTokenTen;
    address public devaddr;
    uint256 public devaddrAmount;
    uint256 public modifyAllocPointPeriod;
    uint256 public lastModifyAllocPointBlock;
    uint256 public totalAllocPoint;
    uint256 public devWithdrawStartBlock;
    uint256 public addpoolfee;
```

```
        uint256 public bonusAllocPointBlock;
        uint256 public minProjectUserCount;

        uint256 public emergencyWithdraw;
        uint256 public constant MINLPTOKEN_AMOUNT = 10;
        uint256 public constant PERSHARERATE = 1000000000000;
        PoolInfo[] public poolInfo;
        PoolSettingInfo[] public poolSettingInfo;
        TenPoolInfo public tenProjectPool;
        TenPoolInfo public tenUserPool;
        mapping (uint256 => mapping (address => UserInfo)) public userInfo;
        mapping (address => UserInfo) public userInfoUserPool;
        mapping (address => bool) public tenMintRightAddr;

        address public bonusAddr;
        uint256 public bonusPoolFeeRate;

        event AddPool(address indexed user, uint256 indexed pid, uint256 tokenAmount,uint256 lpTenAmount);
        event Deposit(address indexed user, uint256 indexed pid, uint256 amount,uint256 penddingToken,uint256
penddingTen,uint256 freezeTen,uint256 freezeBlocks);
        event DepositFrom(address indexed user, uint256 indexed pid, uint256 amount,address from,uint256
penddingToken,uint256 penddingTen,uint256 freezeTen,uint256 freezeBlocks);
        event MineLPToken(address indexed user, uint256 indexed pid, uint256 penddingToken,uint256
penddingTen,uint256 freezeTen,uint256 freezeBlocks);
        event Withdraw(address indexed user, uint256 indexed pid, uint256 amount,uint256 penddingToken,uint256
penddingTen,uint256 freezeTen,uint256 freezeBlocks);

        event DepositLPTen(address indexed user, uint256 indexed pid, uint256 amount,uint256
penddingTen,uint256 freezeTen,uint256 freezeBlocks);
        event WithdrawLPTen(address indexed user, uint256 indexed pid, uint256 amount,uint256
penddingTen,uint256 freezeTen,uint256 freezeBlocks);
        event MineLPTen(address indexed user, uint256 penddingTen,uint256 freezeTen,uint256 freezeBlocks);
        event EmergencyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);
        event DevWithdraw(address indexed user, uint256 amount);

        constructor(
            TenetToken _ten,
            TenetMine _tenMineCalc,
            IERC20 _lpTen,
            address _devaddr,
            uint256 _allocPointProject,
            uint256 _allocPointUser,
            uint256 _devWithdrawStartBlock,
            uint256 _modifyAllocPointPeriod,
            uint256 _bonusAllocPointBlock,
            uint256 _minProjectUserCount
        ) public {
            ten = _ten;
            tenMineCalc = _tenMineCalc;
            devaddr = _devaddr;
            lpTokenTen = _lpTen;
            tenProjectPool.allocPoint = _allocPointProject;
            tenUserPool.allocPoint = _allocPointUser;
            totalAllocPoint = _allocPointProject.add(_allocPointUser);
            devaddrAmount = 0;
            devWithdrawStartBlock = _devWithdrawStartBlock;
            addpoolfee = 0;
            emergencyWithdraw = 0;
            modifyAllocPointPeriod = _modifyAllocPointPeriod;
            lastModifyAllocPointBlock = tenMineCalc.startBlock();
            bonusAllocPointBlock = _bonusAllocPointBlock;
            minProjectUserCount = _minProjectUserCount;
            bonusAddr = msg.sender;
            bonusPoolFeeRate = 0;
        }
        modifier onlyMinter() {
            require(tenMintRightAddr[msg.sender] == true, "onlyMinter: caller is no right to mint");
            _;
        }
        modifier onlyNotEmergencyWithdraw() {
            require(emergencyWithdraw == 0, "onlyNotEmergencyWithdraw: emergencyWithdraw now");
            _;
        }
        function poolLength() external view returns (uint256) {
            return poolInfo.length;
        }
        function set_basicInfo(TenetToken _ten,IERC20 _lpTokenTen,TenetMine _tenMineCalc,uint256
_devWithdrawStartBlock,uint256 _bonusAllocPointBlock,uint256 _minProjectUserCount) public onlyOwner {
            ten = _ten;
            lpTokenTen = _lpTokenTen;
            tenMineCalc = _tenMineCalc;
            devWithdrawStartBlock = _devWithdrawStartBlock;
            bonusAllocPointBlock = _bonusAllocPointBlock;
            minProjectUserCount = _minProjectUserCount;
        }
        function set_bonusInfo(address _bonusAddr,uint256 _addpoolfee,uint256 _bonusPoolFeeRate) public
onlyOwner {
```

```
        bonusAddr = _bonusAddr;
        addpoolfee = _addpoolfee;
        bonusPoolFeeRate = _bonusPoolFeeRate;
    }
    function set_tenMintRightAddr(address _addr,bool isHaveRight) public onlyOwner {
        tenMintRightAddr[_addr] = isHaveRight;
    }
    function tenMint(address _toAddr,uint256 _amount) public onlyMinter {
        if(_amount > 0){
            ten.mint(_toAddr, _amount);
            devaddrAmount = devaddrAmount.add(_amount.div(10));
        }
    }
/*  function set_tenetToken(TenetToken _ten) public onlyOwner {
        ten = _ten;
    } */
    function set_tenNewOwner(address _tenNewOwner) public onlyOwner {
        ten.transferOwnership(_tenNewOwner);
    }
/*  function set_tenetLPToken(IERC20 _lpTokenTen) public onlyOwner {
        lpTokenTen = _lpTokenTen;
    }
    function set_tenetMine(TenetMine _tenMineCalc) public onlyOwner {
        tenMineCalc = _tenMineCalc;
    } */
    function set_emergencyWithdraw(uint256 _emergencyWithdraw) public onlyOwner {
        emergencyWithdraw = _emergencyWithdraw;
    }
/*  function set_addPoolFee(uint256 _addpoolfee) public onlyOwner {
        addpoolfee = _addpoolfee;
    }
    function set_devWithdrawStartBlock(uint256 _devWithdrawStartBlock) public onlyOwner {
        devWithdrawStartBlock = _devWithdrawStartBlock;
    } */
    function set_allocPoint(uint256 _allocPointProject,uint256 _allocPointUser,uint256
_modifyAllocPointPeriod) public onlyOwner {
        _minePoolTen(tenProjectPool);
        _minePoolTen(tenUserPool);
        tenProjectPool.allocPoint = _allocPointProject;
        tenUserPool.allocPoint = _allocPointUser;
        modifyAllocPointPeriod = _modifyAllocPointPeriod;
        totalAllocPoint = _allocPointProject.add(_allocPointUser);
        require(totalAllocPoint > 0, "set_allocPoint: Alloc Point invalid");
    }
/*  function set_bonusAllocPointBlock(uint256 _bonusAllocPointBlock) public onlyOwner {
        bonusAllocPointBlock = _bonusAllocPointBlock;
    }
    function set_minProjectUserCount(uint256 _minProjectUserCount) public onlyOwner {
        minProjectUserCount = _minProjectUserCount;
    } */
    function add(address _lpToken,
                 address _tokenAddr,
                 uint256 _tokenAmount,
                 uint256 _startBlock,
                 uint256 _endBlockOffset,
                 uint256 _tokenPerBlock,
                 uint256 _tokenBonusEndBlockOffset,
                 uint256 _tokenBonusMultipler,
                 Uint256 _lpTenAmount) public onlyNotEmergencyWithdraw {// knownSec // Add liquid mine pool
        if(_startBlock == 0){
            _startBlock = block.number;
        }
        require(block.number <= _startBlock, "add: startBlock invalid");
        require(_endBlockOffset >= _tokenBonusEndBlockOffset, "add: bonusEndBlockOffset invalid");

require(tenMineCalc.getMultiplier(_startBlock, _startBlock.add(_endBlockOffset), _startBlock.add(_endBlockOffse
t), _startBlock.add(_tokenBonusEndBlockOffset), _tokenBonusMultipler).mul(_tokenPerBlock)  <= _tokenAmount,
"add: token amount invalid");
        require(_tokenAmount > 0, "add: tokenAmount invalid");
        IERC20(_tokenAddr).safeTransferFrom(msg.sender,address(this), _tokenAmount);
        if(addpoolfee > 0){
            IERC20(address(ten)).safeTransferFrom(msg.sender,address(this), addpoolfee);
            if(bonusPoolFeeRate > 0){
                IERC20(address(ten)).safeTransfer(bonusAddr,
addpoolfee.mul(bonusPoolFeeRate).div(10000));
                if(bonusPoolFeeRate < 10000){

ten.burn(address(this),addpoolfee.sub(addpoolfee.mul(bonusPoolFeeRate).div(10000)));
                }
            }else{
                ten.burn(address(this),addpoolfee);
            }
        }
        uint256 pid = poolInfo.length;
        PoolSettingInfo. Push (PoolSettingInfo({// knownSec // Add mine pool Settings
                lpToken: _lpToken,
                tokenAddr: _tokenAddr,
```

```
                projectAddr: msg.sender,
                tokenAmount: _tokenAmount,
                startBlock: _startBlock,
                endBlock: _startBlock.add(_endBlockOffset),
                tokenPerBlock: _tokenPerBlock,
                tokenBonusEndBlock: _startBlock.add(_tokenBonusEndBlockOffset),
                tokenBonusMultipler: _tokenBonusMultipler
            }));
        PoolInfo.push (poolInfo ({// knownSec // Add mine pool information
            lastRewardBlock: block.number > _startBlock ?       block.number : _startBlock,
            accTokenPerShare: 0,
            accTenPerShare: 0,
            lpTokenTotalAmount: 0,
            userCount: 0,
            amount: 0,
            rewardTenDebt: 0,
            mineTokenAmount: 0
        }));
        if(_lpTenAmount>=MINLPTOKEN_AMOUNT){
            depositTenByProject(pid,_lpTenAmount);
        }
        emit AddPool(msg.sender, pid, _tokenAmount,_lpTenAmount);
    }
    function updateAllocPoint() public {
        if(lastModifyAllocPointBlock.add(modifyAllocPointPeriod) <= block.number){
            uint256                                 totalLPTokenAmount                                 =
tenProjectPool.lpTokenTotalAmount.mul(bonusAllocPointBlock.add(1e4)).div(1e4).add(tenUserPool.lpTokenTotal
Amount);
            if(totalLPTokenAmount > 0)
            {
                tenProjectPool.allocPoint                                                              =
tenProjectPool.allocPoint.add(tenProjectPool.lpTokenTotalAmount.mul(1e4).mul(bonusAllocPointBlock.add(1e4))
.div(1e4).div(totalLPTokenAmount)).div(2);
                tenUserPool.allocPoint                                                                 =
tenUserPool.allocPoint.add(tenUserPool.lpTokenTotalAmount.mul(1e4).div(totalLPTokenAmount)).div(2);
                totalAllocPoint = tenProjectPool.allocPoint.add(tenUserPool.allocPoint);
                lastModifyAllocPointBlock = block.number;
                require(totalAllocPoint > 0, "updateAllocPoint: Alloc Point invalid");
            }
        }
    }
    // Update reward variables of the given pool to be up-to-date.
    function _minePoolTen(TenPoolInfo storage tenPool) internal {
        if (block.number <= tenPool.lastRewardBlock) {
            return;
        }
        if (tenPool.lpTokenTotalAmount < MINLPTOKEN_AMOUNT) {
            tenPool.lastRewardBlock = block.number;
            return;
        }
        if(emergencyWithdraw > 0){
            tenPool.lastRewardBlock = block.number;
            return;
        }
        uint256 tenReward = tenMineCalc.calcMineTenReward(tenPool.lastRewardBlock, block.number);
        if(tenReward > 0){
            tenReward = tenReward.mul(tenPool.allocPoint).div(totalAllocPoint);
            devaddrAmount = devaddrAmount.add(tenReward.div(10));
            ten.mint(address(this), tenReward);
            tenPool.accTenPerShare                                                                     =
tenPool.accTenPerShare.add(tenReward.mul(PERSHARERATE).div(tenPool.lpTokenTotalAmount));
        }
        tenPool.lastRewardBlock = block.number;
        updateAllocPoint();
    }
    function _withdrawProjectTenPool(PoolInfo storage pool) internal returns (uint256 pending){
        if (pool.amount >= MINLPTOKEN_AMOUNT) {
            pending                                                                                    =
pool.amount.mul(tenProjectPool.accTenPerShare).div(PERSHARERATE).sub(pool.rewardTenDebt);
            if(pending > 0){
                if(pool.userCount == 0){
                    ten.burn(address(this),pending);
                    pending = 0;
                }
                else{
                    if(pool.userCount<minProjectUserCount){
                        uint256                                 newPending                             =
pending.mul(bonusAllocPointBlock.mul(pool.userCount).div(minProjectUserCount).add(1e4)).div(bonusAllocPoi
ntBlock.add(1e4));
                        if(pending.sub(newPending) > 0){
                            ten.burn(address(this),pending.sub(newPending));
                        }
                        pending = newPending;
                    }
                    pool.accTenPerShare                                                                =
pool.accTenPerShare.add(pending.mul(PERSHARERATE).div(pool.lpTokenTotalAmount));
                }
```

```
                }
            }
        }
        function _updateProjectTenPoolAmount(PoolInfo storage pool,uint256 _amount,uint256 amountType) internal{
            if(amountType == 1){
                lpTokenTen.safeTransferFrom(msg.sender, address(this), _amount);
                tenProjectPool.lpTokenTotalAmount = tenProjectPool.lpTokenTotalAmount.add(_amount);
                pool.amount = pool.amount.add(_amount);
            }else if(amountType == 2){
                pool.amount = pool.amount.sub(_amount);
                lpTokenTen.safeTransfer(address(msg.sender), _amount);
                tenProjectPool.lpTokenTotalAmount = tenProjectPool.lpTokenTotalAmount.sub(_amount);
            }
            pool.rewardTenDebt = pool.amount.mul(tenProjectPool.accTenPerShare).div(PERSHARERATE);
        }
        Function DepositEnByProject (UINT256_PID, UINT256_Amount) public OnlyNotemergencywithdraw {/// DepositEnByProject (UINT256_PID, UINT256_Amount
            require(_amount > 0, "depositTenByProject: lpamount not good");
            PoolInfo storage pool = poolInfo[_pid];
            PoolSettingInfo storage poolSetting = poolSettingInfo[_pid];
            require(poolSetting.projectAddr == msg.sender, "depositTenByProject: not good");
            _minePoolTen(tenProjectPool);
            _withdrawProjectTenPool(pool);
            _updateProjectTenPoolAmount(pool, _amount,1);
            Emit DepositLPTen (MSG. The sender, 1, _amount, 0, 0).
        }

        A function descriptive TenbyProject (UINT256_PID, UINT256_Amount) public {/// knownSec // Specifies a pool extract TEN
            PoolInfo storage pool = poolInfo[_pid];
            PoolSettingInfo storage poolSetting = poolSettingInfo[_pid];
            require(poolSetting.projectAddr == msg.sender, "withdrawTenByProject: not good");
            require(pool.amount >= _amount, "withdrawTenByProject: not good");
            if(emergencyWithdraw == 0){
                _minePoolTen(tenProjectPool);
                _withdrawProjectTenPool(pool);
            }else{
                if(pool.lastRewardBlock < poolSetting.endBlock){
                    if(poolSetting.tokenAmount.sub(pool.mineTokenAmount) > 0){

IERC20(poolSetting.tokenAddr).safeTransfer(poolSetting.projectAddr,poolSetting.tokenAmount.sub(pool.mineTokenAmount));
                    }
                }
            }
            if(_amount > 0){
                _updateProjectTenPoolAmount(pool, _amount,2);
            }
            Emit WithdrawLPTen (MSG. The sender, 1, _amount, 0, 0).
        }

        Function updatePoolUserInfo(uint256 acctenperShare,UserInfo storage user,uint256 _freezeBlocks,uint256 _freezeTen,uint256 _amount,uint256 _amountType) internal {/// knownSec // update user information
            if(_amountType == 1){
                user.amount = user.amount.add(_amount);
            }else if(_amountType == 2){
                user.amount = user.amount.sub(_amount);
            }
            user.rewardTenDebt = user.amount.mul(accTenPerShare).div(PERSHARERATE);
            user.lastBlockNumber = block.number;
            user.freezeBlocks = _freezeBlocks;
            user.freezeTen = _freezeTen;
        }
        Function calcFreezeTen(UserInfo storage user,uint256 accTenPerShare) internal view returns (uint256 pendingTen,uint256 freezeBlocks,uint256 freezeTen){///knownsec
            pendingTen = user.amount.mul(accTenPerShare).div(PERSHARERATE).sub(user.rewardTenDebt);
            uint256 blockNow = 0;
            if(pendingTen>0){
                if(user.lastBlockNumber<tenMineCalc.startBlock()){
                    blockNow = block.number.sub(tenMineCalc.startBlock());
                }else{
                    blockNow = block.number.sub(user.lastBlockNumber);
                }
            }else{
                if(user.freezeTen > 0){
                    blockNow = block.number.sub(user.lastBlockNumber);
                }
            }
            uint256 periodBlockNumer = tenMineCalc.subBlockNumerPeriod();
            freezeBlocks = blockNow.add(user.freezeBlocks);
            if(freezeBlocks <= periodBlockNumer){
                freezeTen = pendingTen.add(user.freezeTen);
                pendingTen = 0;
            }else{
                if(pendingTen == 0){
```

```
                freezeBlocks = 0;
                freezeTen = 0;
                pendingTen = user.freezeTen;
            }else{
                freezeTen = pendingTen.add(user.freezeTen).mul(periodBlockNumer).div(freezeBlocks);
                pendingTen = pendingTen.add(user.freezeTen).sub(freezeTen);
                freezeBlocks = periodBlockNumer;
            }
        }
    }
    function _withdrawUserTenPool(address userAddr,UserInfo storage user) internal returns (uint256 pendingTen,uint256 freezeBlocks,uint256 freezeTen){
        (pendingTen,freezeBlocks,freezeTen) = _calcFreezeTen(user,tenUserPool.accTenPerShare);
        safeTenTransfer(userAddr, pendingTen);
    }
    Function DepositEnByUser (uint256 _amount) public OnlyNotemergencywithdraw {/// DepositEnByUser (uint256 _amount) public OnlyNotemergencywithdraw {///
        require(_amount > 0, "depositTenByUser: lpamount not good");
        UserInfo storage user = userInfoUserPool[msg.sender];
        _minePoolTen(tenUserPool);
        (uint256 pending,uint256 freezeBlocks,uint256 freezeTen) = _withdrawUserTenPool(msg.sender,user);
        lpTokenTen.safeTransferFrom(address(msg.sender), address(this), _amount);
        _updatePoolUserInfo(tenUserPool.accTenPerShare,user,freezeBlocks,freezeTen, _amount,1);
        tenUserPool.lpTokenTotalAmount = tenUserPool.lpTokenTotalAmount.add(_amount);
        emit DepositLPTen(msg.sender, 2, _amount,pending,freezeTen,freezeBlocks);
    }

    Function descriptive tenbyuser (uint256 _amount) public {//knownsec// extract ten token
        require(_amount > 0, "withdrawTenByUser: lpamount not good");
        UserInfo storage user = userInfoUserPool[msg.sender];
        require(user.amount >= _amount, "withdrawTenByUser: not good");
        if(emergencyWithdraw == 0){
            _minePoolTen(tenUserPool);
            (uint256 pending,uint256 freezeBlocks,uint256 freezeTen) = _withdrawUserTenPool(msg.sender,user);
            _updatePoolUserInfo(tenUserPool.accTenPerShare,user,freezeBlocks,freezeTen, _amount,2);
            emit WithdrawLPTen(msg.sender, 2, _amount,pending,freezeTen,freezeBlocks);
        }else{

            _updatePoolUserInfo(tenUserPool.accTenPerShare,user,user.freezeBlocks,user.freezeTen, _amount,2);
            emit WithdrawLPTen(msg.sender, 2, _amount,0,user.freezeTen,user.freezeBlocks);
        }
        tenUserPool.lpTokenTotalAmount = tenUserPool.lpTokenTotalAmount.sub(_amount);
        lpTokenTen.safeTransfer(address(msg.sender), _amount);
    }
    Function mineLPTen() public onlyNotEmergencyWithdraw {// knownSec //
        _minePoolTen(tenUserPool);
        UserInfo storage user = userInfoUserPool[msg.sender];
        (uint256 pending,uint256 freezeBlocks,uint256 freezeTen) = _withdrawUserTenPool(msg.sender,user);
        _updatePoolUserInfo (tenUserPool accTenPerShare, user, freezeBlocks freezeTen, 0, 0).
        emit MineLPTen(msg.sender,pending,freezeTen,freezeBlocks);
    }
    function depositTenByUserFrom(address _from,uint256 _amount) public onlyNotEmergencyWithdraw {
        require(_amount > 0, "depositTenByUserFrom: lpamount not good");
        UserInfo storage user = userInfoUserPool[_from];
        _minePoolTen(tenUserPool);
        (uint256 pending,uint256 freezeBlocks,uint256 freezeTen) = _withdrawUserTenPool(_from,user);
        lpTokenTen.safeTransferFrom(address(msg.sender), address(this), _amount);
        _updatePoolUserInfo(tenUserPool.accTenPerShare,user,freezeBlocks,freezeTen, _amount,1);
        tenUserPool.lpTokenTotalAmount = tenUserPool.lpTokenTotalAmount.add(_amount);
        emit DepositLPTen(_from, 2, _amount,pending,freezeTen,freezeBlocks);
    }
    Function _minepoolToken (PoolInfo storage pool,PoolSettingInfo storage poolSetting) internal {//knownsec//
        if (block.number <= pool.lastRewardBlock) {
            return;
        }
        if (pool.lpTokenTotalAmount >= MINLPTOKEN_AMOUNT) {
            uint256 multiplier = tenMineCalc.getMultiplier(pool.lastRewardBlock, block.number,poolSetting.endBlock,poolSetting.tokenBonusEndBlock,poolSetting.tokenBonusMultipler);
            if(multiplier > 0){
                uint256 tokenReward = multiplier.mul(poolSetting.tokenPerBlock);
                pool.mineTokenAmount = pool.mineTokenAmount.add(tokenReward);
                pool.accTokenPerShare = pool.accTokenPerShare.add(tokenReward.mul(PERSHARERATE).div(pool.lpTokenTotalAmount));
            }
        }
        if(pool.lastRewardBlock < poolSetting.endBlock){
            if(block.number >= poolSetting.endBlock){
                if(poolSetting.tokenAmount.sub(pool.mineTokenAmount) > 0){

IERC20(poolSetting.tokenAddr).safeTransfer(poolSetting.projectAddr,poolSetting.tokenAmount.sub(pool.mineTokenAmount));
                }
            }
        }
        pool.lastRewardBlock = block.number;
```

```
            _minePoolTen(tenProjectPool);
            _withdrawProjectTenPool(pool);
            _updateProjectTenPoolAmount (pool, 0, 0);
        }
        function withdrawTokenPool(address userAddr,PoolInfo storage pool,UserInfo storage user,PoolSettingInfo storage poolSetting)
            internal returns (uint256 pendingToken,uint256 pendingTen,uint256 freezeBlocks,uint256 freezeTen){
            if (user.amount >= MINLPTOKEN_AMOUNT) {
                pendingToken = user.amount.mul(pool.accTokenPerShare).div(PERSHARERATE).sub(user.rewardTokenDebt);
                if(pendingToken > 0){
                    IERC20(poolSetting.tokenAddr).safeTransfer(userAddr, pendingToken);
                }
                (pendingTen,freezeBlocks,freezeTen) = _calcFreezeTen(user,pool.accTenPerShare);
                safeTenTransfer(userAddr, pendingTen);
            }
        }
        function _updateTokenPoolUser(uint256 accTokenPerShare,uint256 accTenPerShare,UserInfo storage user,uint256 _freezeBlocks,uint256 _freezeTen,uint256 _amount,uint256 _amountType)
            internal {
            _updatePoolUserInfo(accTenPerShare,user,_freezeBlocks,_freezeTen,_amount,_amountType);
            user.rewardTokenDebt = user.amount.mul(accTokenPerShare).div(PERSHARERATE);
        }
        Function depositLPToken(uint256_pid, uint256_amount) public onlyNotEmergencyWithdraw {//knownsc// DepositLPToken (uint256_pid, uint256_amount) public onlyNotEmergencyWithdraw {//knownsc//
            require(_amount > 0, "depositLPToken: lpamount not good");
            PoolInfo storage pool = poolInfo[_pid];
            PoolSettingInfo storage poolSetting = poolSettingInfo[_pid];
            UserInfo storage user = userInfo[_pid][msg.sender];
            _minePoolToken(pool,poolSetting);
            (uint256 pendingToken,uint256 pendingTen,uint256 freezeBlocks,uint256 freezeTen) = _withdrawTokenPool(msg.sender,pool,user,poolSetting);
            if (user.amount < MINLPTOKEN_AMOUNT) {
                pool.userCount = pool.userCount.add(1);
            }
            IERC20(poolSetting.lpToken).safeTransferFrom(address(msg.sender), address(this), _amount);
            pool.lpTokenTotalAmount = pool.lpTokenTotalAmount.add(_amount);

            _updateTokenPoolUser(pool.accTokenPerShare,pool.accTenPerShare,user,freezeBlocks,freezeTen,_amount,1);
            emit Deposit(msg.sender, _pid, _amount,pendingToken,pendingTen,freezeTen,freezeBlocks);
        }

        A function withdrawing lpToken (uint256 _pid, uint256 _amount) public {// knownSec // Extract the liquid token from the specified pool
            require(_amount > 0, "withdrawLPToken: lpamount not good");
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][msg.sender];
            PoolSettingInfo storage poolSetting = poolSettingInfo[_pid];
            require(user.amount >= _amount, "withdrawLPToken: not good");
            if(emergencyWithdraw == 0){
                _minePoolToken(pool,poolSetting);
                (uint256 pendingToken,uint256 pendingTen,uint256 freezeBlocks,uint256 freezeTen) = _withdrawTokenPool(msg.sender,pool,user,poolSetting);

            _updateTokenPoolUser(pool.accTokenPerShare,pool.accTenPerShare,user,freezeBlocks,freezeTen,_amount,2);
                emit Withdraw(msg.sender, _pid, _amount,pendingToken,pendingTen,freezeTen,freezeBlocks);
            }else{

            _updateTokenPoolUser(pool.accTokenPerShare,pool.accTenPerShare,user,user.freezeBlocks,user.freezeTen,_amount,2);
                Emit Withdraw (MSG) sender, _pid, _amount, 0, 0, user. FreezeTen, user. FreezeBlocks);
            }
            IERC20(poolSetting.lpToken).safeTransfer(address(msg.sender), _amount);
            pool.lpTokenTotalAmount = pool.lpTokenTotalAmount.sub(_amount);
            if(user.amount < MINLPTOKEN_AMOUNT){
                pool.userCount = pool.userCount.sub(1);
            }
        }

        Function minelpToken (uint256 _pid) public onlyNotEmergencyWithdraw {// knownSec //
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][msg.sender];
            PoolSettingInfo storage poolSetting = poolSettingInfo[_pid];
            _minePoolToken(pool,poolSetting);
            (uint256 pendingToken,uint256 pendingTen,uint256 freezeBlocks,uint256 freezeTen) = _withdrawTokenPool(msg.sender,pool,user,poolSetting);
            _updateTokenPoolUser (pool. AccTokenPerShare, pool accTenPerShare, user, freezeBlocks, freezeTen, 0, 0).
            emit MineLPToken(msg.sender, _pid, pendingToken,pendingTen,freezeTen,freezeBlocks);
        }

        function depositLPTokenFrom(address _from,uint256 _pid, uint256 _amount) public onlyNotEmergencyWithdraw {
            require(_amount > 0, "depositLPTokenFrom: lpamount not good");
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][_from];
```

```
            PoolSettingInfo storage poolSetting = poolSettingInfo[_pid];
            _minePoolToken(pool,poolSetting);
            (uint256 pendingToken,uint256 pendingTen,uint256 freezeBlocks,uint256 freezeTen) =
_withdrawTokenPool(_from,pool,user,poolSetting);
            if (user.amount < MINLPTOKEN_AMOUNT) {
                pool.userCount = pool.userCount.add(1);
            }
            IERC20(poolSetting.lpToken).safeTransferFrom(msg.sender, address(this), _amount);
            pool.lpTokenTotalAmount = pool.lpTokenTotalAmount.add(_amount);

_updateTokenPoolUser(pool.accTokenPerShare,pool.accTenPerShare,user,freezeBlocks,freezeTen,_amount,1);
            emit DepositFrom(_from, _pid, _amount,msg.sender,pendingToken,pendingTen,freezeTen,freezeBlocks);
        }

        Function dev(address _devaddr) public {// knownSec // update developer address
            require(msg.sender == devaddr, "dev: wut?" );
            devaddr = _devaddr;
        }

        Function devWithdraw(uint256 _amount) public {// knownSec //
            require(block.number >= devWithdrawStartBlock, "devWithdraw: start Block invalid");
            require(msg.sender == devaddr, "devWithdraw: devaddr invalid");
            require(devaddrAmount >= _amount, "devWithdraw: amount invalid");
            ten.mint(devaddr, _amount);
            devaddrAmount = devaddrAmount.sub(_amount);
            emit DevWithdraw(msg.sender, _amount);
        }

        function safeTenTransfer(address _to, uint256 _amount) internal {
            if(_amount > 0){
                uint256 bal = ten.balanceOf(address(this));
                if (_amount > bal) {
                    if(bal > 0){
                        IERC20(address(ten)).safeTransfer(_to, bal);
                    }
                } else {
                    IERC20(address(ten)).safeTransfer(_to, _amount);
                }
            }
        }
}
```

TenetMine.sol

```
// SPDX-License-Identifier: MIT

Pragma solidity 0.6.12;


import "./openzeppelin/contracts/token/ERC20/IERC20.sol";
import "./openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "./openzeppelin/contracts/utils/EnumerableSet.sol";
import "./openzeppelin/contracts/math/SafeMath.sol";
import "./openzeppelin/contracts/access/Ownable.sol";
contract TenetMine is Ownable {
        using SafeMath for uint256;
        struct MinePeriodInfo {
            uint256 tenPerBlockPeriod;
            uint256 totalTenPeriod;
        }
        uint256 public bonusEndBlock;
        uint256 public bonus_multiplier;
        uint256 public bonusTenPerBlock;
        uint256 public startBlock;
        uint256 public endBlock;
        uint256 public subBlockNumerPeriod;
        uint256 public totalSupply;
        MinePeriodInfo[] public allMinePeriodInfo;

        constructor(
            uint256 _startBlock,
            uint256 _bonusEndBlockOffset,
            uint256 _bonus_multiplier,
            uint256 _bonusTenPerBlock,
            uint256 _subBlockNumerPeriod,
            uint256[] memory _tenPerBlockPeriod
        ) public {
            startBlock = _startBlock>0 ?       startBlock : block.number + 1;
            bonusEndBlock = startBlock.add(_bonusEndBlockOffset);
            bonus_multiplier = _bonus_multiplier;
            bonusTenPerBlock = _bonusTenPerBlock;
            subBlockNumerPeriod = _subBlockNumerPeriod;
            totalSupply = bonusEndBlock.sub(startBlock).mul(bonusTenPerBlock).mul(bonus_multiplier);
            for (uint256 i = 0;      i < _tenPerBlockPeriod.length;      i++) {
                allMinePeriodInfo.push(MinePeriodInfo({
                    tenPerBlockPeriod: _tenPerBlockPeriod[i],
```

```
                totalTenPeriod: totalSupply
            }));
            totalSupply = totalSupply.add(subBlockNumerPeriod.mul(_tenPerBlockPeriod[i]));
        }
        endBlock = bonusEndBlock.add(subBlockNumerPeriod.mul(_tenPerBlockPeriod.length));
    }
    Function set_startBlock(uint256 _startBlock) public onlyOwner {// knownSec // set_startBlock
        require(block.number < _startBlock, "set_startBlock: startBlock invalid");
        uint256 bonusEndBlockOffset = bonusEndBlock.sub(startBlock);
        startBlock = _startBlock>0 ?    startBlock : block.number + 1;
        bonusEndBlock = startBlock.add(bonusEndBlockOffset);
        endBlock = bonusEndBlock.add(subBlockNumerPeriod.mul(allMinePeriodInfo.length));
    }
    function getMinePeriodCount() public view returns (uint256) {
        return allMinePeriodInfo.length;
    }
    Function calcminetenReply (uint256 _from, uint256 _to) public view returns (uint256) {// knownSec returns
(uint256 _to
        if(_from < startBlock){
            _from = startBlock;
        }
        if(_from >= endBlock){
            return 0;
        }
        if(_from >= _to){
            return 0;
        }
        uint256 mineFrom = calcTotalMine(_from);
        uint256 mineTo= calcTotalMine(_to);
        return mineTo.sub(mineFrom);
    }
    Function calcTotalMine(uint256 _to) public view returns (uint256) {// knownSec returns ()
        if(_to <= startBlock){
            totalMine = 0;
        }else if(_to <= bonusEndBlock){
            totalMine = _to.sub(startBlock).mul(bonusTenPerBlock).mul(bonus_multiplier);
        }else if(_to < endBlock){
            uint256 periodIndex = _to.sub(bonusEndBlock).div(subBlockNumerPeriod);
            uint256 periodBlock = _to.sub(bonusEndBlock).mod(subBlockNumerPeriod);
            MinePeriodInfo memory minePeriodInfo = allMinePeriodInfo[periodIndex];
            uint256 curMine = periodBlock.mul(minePeriodInfo.tenPerBlockPeriod);
            totalMine = curMine.add(minePeriodInfo.totalTenPeriod);
        }else{
            totalMine = totalSupply;
        }
    }
    // Return reward multiplier over the given _from to _to block.
    Function getMultiplier (uint256 _from, uint256 _to, uint256 _end, uint256 _tokenBonusEndBlock, uint256
_tokenBonusMultipler) public pure returns (uint256) {// / knownsec calculate according to the number of blocks / /
factors
        if(_to > _end){
            _to = _end;
        }
        if(_from> _end){
            return 0;
        }else if (_to <= _tokenBonusEndBlock) {
            return _to.sub(_from).mul(_tokenBonusMultipler);
        } else if (_from >= _tokenBonusEndBlock) {
            return _to.sub(_from);
        } else {
            return
_tokenBonusEndBlock.sub(_from).mul(_tokenBonusMultipler).add(_to.sub(_tokenBonusEndBlock));
        }
    }
}
```

TenetProxy.sol

```
// SPDX-License-Identifier: MIT

Pragma solidity 0.6.12;

import "./openzeppelin/contracts/token/ERC20/ERC20.sol";
import "./openzeppelin/contracts/token/ERC20/IERC20.sol";
import "./openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "./openzeppelin/contracts/utils/EnumerableSet.sol";
import "./openzeppelin/contracts/math/SafeMath.sol";
import "./openzeppelin/contracts/access/Ownable.sol";
import "./uniswapv2/IUniswapV2Pair.sol";
import "./uniswapv2/IUniswapV2Factory.sol";
import "./TenetMine.sol";
import "./Tenet.sol";

contract TenetProxy is Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
```

```
        uint256 public constant MINLPTOKEN_AMOUNT = 10;
        uint public constant MINIMUM_LIQUIDITY = 10**3;
        uint256 public constant PERSHARERATE = 1000000000000;

        Tenet public tenet;
        TenetMine public tenetmine;
        constructor(Tenet _tenet) public {
            tenet = _tenet;
            tenetmine = tenet.tenMineCalc();
        }
        function set_tenet(Tenet _tenet) public onlyOwner {
            tenet = _tenet;
            tenetmine = tenet.tenMineCalc();
        }
        function getPoolAllInfo(uint256 _pid) public view returns (address[3] memory retData1,uint256[6] memory
retData2,uint256[8] memory retData3) {
            (retData1) = getPoolSettingInfo1(_pid);
            (retData2) = getPoolSettingInfo2(_pid);
            (retData3) = getPoolInfo(_pid);
        }
        function getPoolSettingInfo1(uint256 _pid) public view returns (address[3] memory retData1) {
            (retData1[0],retData1[1],retData1[2],,,,,,) = tenet.poolSettingInfo(_pid);
        }
        function getPoolSettingInfo2(uint256 _pid) public view returns (uint256[6] memory retData2) {
            (,,,retData2[0],retData2[1],retData2[2],retData2[3],retData2[4],retData2[5])                    =
tenet.poolSettingInfo(_pid);
        }
        function getPoolInfo(uint256 _pid) public view returns (uint256[8] memory retData3) {
            (retData3[0],retData3[1],retData3[2],retData3[3],retData3[4],retData3[5],retData3[6],retData3[7])
= tenet.poolInfo(_pid);
        }

        function getPendingTenByProject(uint _pid) public view returns (uint256) {
            ( , ,uint256[8] memory retData3) = getPoolAllInfo(_pid);
            if(retData3[1] < MINLPTOKEN_AMOUNT){
                return 0;
            }
            if(retData3[5] < MINLPTOKEN_AMOUNT){
                return 0;
            }
            uint256[4] memory tenPoolInfo;
            (tenPoolInfo[0],tenPoolInfo[1],tenPoolInfo[2],tenPoolInfo[3]) = tenet.tenProjectPool();
            if(tenPoolInfo[3] < MINLPTOKEN_AMOUNT){
                return 0;
            }
            if (block.number > tenPoolInfo[0] && retData3[5] != 0) {
                uint256 tenReward = tenetmine.calcMineTenReward(tenPoolInfo[0], block.number);
                tenReward = tenReward.mul(tenPoolInfo[2]).div(tenet.totalAllocPoint());
                tenPoolInfo[1] = tenPoolInfo[1].add(tenReward.mul(1e12).div(tenPoolInfo[3]));
            }
            return retData3[5].mul(tenPoolInfo[1]).div(1e12).sub(retData3[6]);
        }
        function _calcFreezeTen(uint256[6] memory userInfo,uint256 accTenPerShare) internal view returns
(uint256 pendingTen,uint256 freezeBlocks,uint256 freezeTen){
            pendingTen = userInfo[0].mul(accTenPerShare).div(PERSHARERATE).sub(userInfo[2]);
            uint256 blockNow = block.number.sub(userInfo[3]);
            uint256 periodBlockNumer = tenetmine.subBlockNumerPeriod();
            freezeBlocks = blockNow.add(userInfo[4]);
            if(freezeBlocks <= periodBlockNumer){
                freezeTen = pendingTen.add(userInfo[5]);
                pendingTen = 0;
            }else{
                if(pendingTen == 0){
                    freezeBlocks = 0;
                    freezeTen = 0;
                    pendingTen = userInfo[5];
                }else{
                    freezeTen = pendingTen.add(userInfo[5]).mul(periodBlockNumer).div(freezeBlocks);
                    pendingTen = pendingTen.add(userInfo[5]).sub(freezeTen);
                    freezeBlocks = periodBlockNumer;
                }
            }
        }

        function getPendingTenByUser(address _user) public view returns (uint256,uint256,uint256) {
            uint256[6] memory userInfo;
            (userInfo[0],userInfo[1],userInfo[2],userInfo[3],userInfo[4],userInfo[5])                       =
tenet.userInfoUserPool(_user);
            if(userInfo[0] < MINLPTOKEN_AMOUNT){
                if(block.number.sub(userInfo[3])>tenetmine.subBlockNumerPeriod()){
                    Return (the userInfo [5], 0, 0).
                }else{
                    Return (0, 0, the userInfo [5]);
                }
            }
            uint256[4] memory tenPoolInfo;
            (tenPoolInfo[0],tenPoolInfo[1],tenPoolInfo[2],tenPoolInfo[3]) = tenet.tenUserPool();
```

```
            if(tenPoolInfo[3] < MINLPTOKEN_AMOUNT){
                if(block.number.sub(userInfo[3])>tenetmine.subBlockNumerPeriod()){
                    Return (the userInfo [5], 0, 0).
                }else{
                    Return (0, 0, the userInfo [5]);
                }
            }
            if (block.number > tenPoolInfo[0]) {
                uint256 tenReward = tenetmine.calcMineTenReward(tenPoolInfo[0], block.number);
                tenReward = tenReward.mul(tenPoolInfo[2]).div(tenet.totalAllocPoint());
                tenPoolInfo[1] = tenPoolInfo[1].add(tenReward.mul(1e12).div(tenPoolInfo[3]));
            }
            (uint256 pendingTen,uint256 freezeBlocks,uint256 freezeTen) =
_calcFreezeTen(userInfo,tenPoolInfo[1]);
            return (pendingTen,freezeBlocks,freezeTen);
        }


        Function getPendingTen(uint256,uint256,uint256) public view returns (uint256,uint256,uint256)
{//knownsec// getPendingTen
            uint256[6] memory userInfo;
            (userInfo[0], ,userInfo[2],userInfo[3],userInfo[4],userInfo[5]) = tenet.userInfo(_pid,_user);
            if(userInfo[0] < MINLPTOKEN_AMOUNT){
                if(block.number.sub(userInfo[3])>tenetmine.subBlockNumerPeriod()){
                    Return (the userInfo [5], 0, 0).
                }else{
                    Return (0, 0, the userInfo [5]);
                }
            }
            ( , ,uint256[8] memory retData3) = getPoolAllInfo(_pid);
            if(retData3[1] < MINLPTOKEN_AMOUNT){
                if(block.number.sub(userInfo[3])>tenetmine.subBlockNumerPeriod()){
                    Return (the userInfo [5], 0, 0).
                }else{
                    Return (0, 0, the userInfo [5]);
                }
            }
            uint256 pending = getPendingTenByProject(_pid);
            retData3[3] = retData3[3].add(pending.mul(1e12).div(retData3[1]));
            (uint256 pendingTen,uint256 freezeBlocks,uint256 freezeTen) = _calcFreezeTen(userInfo,retData3[3]);
            return (pendingTen,freezeBlocks,freezeTen);
        }


        Function getPendingToken(uint256 _pid, address _user) public view returns (uint256) {// knownSec // Get
the pending token
            ( ,uint256[6] memory retData2,uint256[8] memory retData3) = getPoolAllInfo(_pid);
            if(retData3[1] < MINLPTOKEN_AMOUNT){
                return 0;
            }
            uint256[6] memory userInfo;
            (userInfo[0],userInfo[2], , , , ) = tenet.userInfo(_pid,_user);
            if(userInfo[0] < MINLPTOKEN_AMOUNT){
                return 0;
            }
            if (block.number > retData3[0] && retData3[1] != 0) {
                uint256 tokenReward = retData2[3].mul(tenetmine.getMultiplier(retData3[0],
block.number,retData2[2],retData2[4],retData2[5]));
                retData3[2] = retData3[2].add(tokenReward.mul(1e12).div(retData3[1]));
            }
            return userInfo[0].mul(retData3[2]).div(1e12).sub(userInfo[2]);
        }
        Function calcLiquidity2(address _pairAddr,uint256 _token0Amount,uint256 _token1Amount) public
liquidity = function calcLiquidity2(address _pairAddr,uint256 _token0Amount,uint256 _token1Amount
            uint256 totalSupply = IUniswapV2Pair(_pairAddr).totalSupply();
            (uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(_pairAddr).getReserves();
            if(totalSupply == 0){
                liquidity = sqrt(_token0Amount.mul(_token1Amount)).sub(MINIMUM_LIQUIDITY);
            }else {
                liquidity = min(_token0Amount.mul(totalSupply) / reserve0, _token1Amount.mul(totalSupply) /
reserve1);
            }
        }
        Function calcLiquidity(address _pairAddr,address _tokenAddr,uint256 _tokenAmount) public view returns
(uint256 liquidity) {// knownSec // Calculate the liquidity
            uint256[2] memory tokenAmountOut;
            if(_tokenAddr == IUniswapV2Pair(_pairAddr).token0()){
                (tokenAmountOut[0],tokenAmountOut[1])                                                =
calcTokenXOut(_pairAddr, _tokenAddr, _tokenAmount,0);
            }else if(_tokenAddr == IUniswapV2Pair(_pairAddr).token1()){
                (tokenAmountOut[0],tokenAmountOut[1])                                                =
calcTokenXOut(_pairAddr, _tokenAddr, _tokenAmount,1);
            }else{
                (tokenAmountOut[0],tokenAmountOut[1])                                                =
calcTokensOut(_pairAddr, _tokenAddr, _tokenAmount);
            }
            if(tokenAmountOut[0] == 0){
```

```
                liquidity = 0;
            }else if(tokenAmountOut[0] == 0){
                liquidity = 0;
            }else{
                uint256 totalSupply = IUniswapV2Pair(_pairAddr).totalSupply();
                (uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(_pairAddr).getReserves();
                if(totalSupply == 0){
                    liquidity = sqrt(tokenAmountOut[0].mul(tokenAmountOut[1])).sub(MINIMUM_LIQUIDITY);
                }else{
                    liquidity = min(tokenAmountOut[0].mul(totalSupply) / reserve0, tokenAmountOut[1].mul(totalSupply) / reserve1);
                }
            }
        }

        function getAmountOut(address _pairAddr, address _fromAddr,uint amountIn) public view virtual returns (uint256){
            //require(amountIn > 0, 'getAmountOut: INSUFFICIENT_INPUT_AMOUNT');
            if(amountIn == 0){
                return 0;
            }
            (uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(_pairAddr).getReserves();
            //require(reserve0 > 0 && reserve1 > 0, 'getAmountOut: INSUFFICIENT_LIQUIDITY');
            if(reserve0 == 0){
                return 0;
            }
            if(reserve1 == 0){
                return 0;
            }
            uint amountInWithFee = amountIn.mul(997);
            if(_fromAddr == IUniswapV2Pair(_pairAddr).token0()){
                uint numerator = amountInWithFee.mul(reserve1);
                uint denominator = reserve0.mul(1000).add(amountInWithFee);
                return numerator.div(denominator);
            }else{
                uint numerator = amountInWithFee.mul(reserve0);
                uint denominator = reserve1.mul(1000).add(amountInWithFee);
                return numerator.div(denominator);
            }
        }

        Function getPrice(address _pairAddr, address _fromAddr) public view returns (uint256) {// knownSec //
            (uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(_pairAddr).getReserves();
            if(_fromAddr == IUniswapV2Pair(_pairAddr).token0()){
                return reserve1.mul(1e12).div(reserve0);
            }else{
                return reserve0.mul(1e12).div(reserve1);
            }
        }

        function calcTokensOut(address _pairAddr,address _tokenAddr,uint256 _tokenAmount) public view returns (uint256,uint256) {
            IUniswapV2Factory factory = IUniswapV2Factory(IUniswapV2Pair(_pairAddr).factory());
            //require(address(factory) != address(0), 'calcTokensOut: INSUFFICIENT_PAIRADDR');
            if(address(factory) == address(0)){
                Return (0, 0);
            }
            uint256[8] memory dataAll;
            (dataAll[6], dataAll[7],) = IUniswapV2Pair(_pairAddr).getReserves();
            //require(dataAll[6] > 0, 'calcTokenOut: INSUFFICIENT_RESERVE0');
            //require(dataAll[7] > 0, 'calcTokenOut: INSUFFICIENT_RESERVE1');
            if(dataAll[6] == 0){
                Return (0, 0);
            }
            if(dataAll[7] == 0){
                Return (0, 0);
            }
            address[2] memory allPairAddr;
            allPairAddr[0] = factory.getPair(_tokenAddr,IUniswapV2Pair(_pairAddr).token0());
            //require(allPairAddr[0] != address(0), 'calcToken: INVALID_PAIR0');
            if(allPairAddr[0] == address(0)){
                Return (0, 0);
            }
            dataAll[0] = getPrice(allPairAddr[0],_tokenAddr);
            allPairAddr[1] = factory.getPair(_tokenAddr,IUniswapV2Pair(_pairAddr).token1());
            //require(allPairAddr[1] != address(0), 'calcToken: INVALID_PAIR1');
            if(allPairAddr[1] == address(0)){
                Return (0, 0);
            }
            dataAll[1] = getPrice(allPairAddr[1],_tokenAddr);

            dataAll[2] = _tokenAmount.mul(dataAll[1]).mul(dataAll[6]).div(dataAll[0].mul(dataAll[7]).add(dataAll[1].mul(dataAll[6])));

            dataAll[3] = _tokenAmount.sub(dataAll[2]);
            dataAll[4] = getAmountOut(allPairAddr[0],_tokenAddr,dataAll[2]);
```

```
            dataAll[5] = getAmountOut(allPairAddr[1],_tokenAddr,dataAll[3]);
            return (dataAll[4],dataAll[5]);
        }

        function calcTokenXOut(address _pairAddr,address _tokenAddr,uint256 _tokenAmount,uint256 tokenType)
public view returns (uint256,uint256) {
            IUniswapV2Factory factory = IUniswapV2Factory(IUniswapV2Pair(_pairAddr).factory());
            //require(address(factory) != address(0), 'calcTokenXOut: INSUFFICIENT_PAIRADDR');
            if(address(factory) == address(0)){
                Return (0, 0);
            }
            uint256[5] memory dataAll;
            (dataAll[0], dataAll[1],) = IUniswapV2Pair(_pairAddr).getReserves();
            //require(dataAll[0] > 0, 'calcTokenXOut: INSUFFICIENT_RESERVE0');
            //require(dataAll[1] > 0, 'calcTokenXOut: INSUFFICIENT_RESERVE1');
            if(dataAll[0] == 0){
                Return (0, 0);
            }
            if(dataAll[1] == 0){
                Return (0, 0);
            }
            // (reserv_USDT * amount / (reserv_USDT + reserv_TEN) )
            dataAll[2] = _tokenAmount.div(2);
            dataAll[3] = _tokenAmount.sub(dataAll[2]);
            dataAll[4] = getAmountOut(_pairAddr,_tokenAddr,dataAll[3]);
            if(tokenType == 0){
                return (dataAll[2],dataAll[4]);
            }else{
                return (dataAll[4],dataAll[2]);
            }
        }
        function min(uint x, uint y) internal pure returns (uint z) {
            z = x < y ?     x : y;
        }
        function sqrt(uint y) internal pure returns (uint z) {
            if (y > 3) {
                z = y;
                uint x = y / 2 + 1;
                while (x < z) {
                    z = x;
                    x = (y / x + x) / 2;
                }
            } else if (y != 0) {
                z = 1;
            }
        }
}
```

TenetProxyInner.sol

```
// SPDX-License-Identifier: MIT

Pragma solidity 0.6.12;

import "./openzeppelin/contracts/token/ERC20/ERC20.sol";
import "./openzeppelin/contracts/token/ERC20/IERC20.sol";
import "./openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "./openzeppelin/contracts/utils/EnumerableSet.sol";
import "./openzeppelin/contracts/math/SafeMath.sol";
import "./openzeppelin/contracts/access/Ownable.sol";
import "./uniswapv2/IUniswapV2Pair.sol";
import "./uniswapv2/IUniswapV2Factory.sol";
import "./TenetMine.sol";
import "./Tenet.sol";

contract TenetProxyInner is Ownable {
        using SafeMath for uint256;
        using SafeERC20 for IERC20;

        uint256 public constant MINLPTOKEN_AMOUNT = 10;
        uint256 public constant MINWEALTH_AMOUNT = 1000000000000000000;
        Tenet public tenet;
        TenetMine public tenetmine;
        address public wethAddr;
        address public wusdtAddr;
        IUniswapV2Factory public uniFactory;
        constructor(Tenet _tenet,address _weth,address _wusdt) public {
            tenet = _tenet;
            tenetmine = tenet.tenMineCalc();
            wethAddr = _weth;
            wusdtAddr = _wusdt;
            uniFactory = IUniswapV2Factory(IUniswapV2Pair(address(tenet.lpTokenTen())).factory());
        }
        function set_tenet(Tenet _tenet) public onlyOwner {
            tenet = _tenet;
            tenetmine = tenet.tenMineCalc();
```

```
    }

    Function getTenPoolNewInfo() public view Returns (UINT256 [6] Memory RetDatas1, UINT256 [6] Memory RetDatas2, UINT256 [8] Memory RetDatas3) {// knownSec returns () public view Returns (UINT256 [6] Memory RetDatas2, UINT256 [8] Memory RetDatas3)
        retDatas1 = getTenUserPool();
//(lastRewardBlock,accTenPerShare,allocPoint,lpTokenTotalAmount,totalAllocPoint,newBlockTen);
        retDatas2 = getTenProjectPool();
//(lastRewardBlock,accTenPerShare,allocPoint,lpTokenTotalAmount,totalAllocPoint,newBlockTen);
        retDatas3 = getPoolPriceInfo(address(tenet.lpTokenTen()),address(tenet.ten()));
//(lpSupply,reserve0,reserve1,pricetype,price0,price1,tokeneth,tokenusdt);
    }

    Function getTokenPoolNewInfo(uint256 _pid) public view returns (uint256[8] Memory RetDatas1,uint256[8] Memory RetDatas3) {// knownSec // getTokenPoolNewInfo(uint256 _pid) public view returns (uint256[8] Memory RetDatas3,uint256[8] Memory RetDatas3) {// knownSec //
        retDatas1 = getPoolInfo(_pid);
//(lastRewardBlock,lpTokenTotalAmount,accTokenPerShare,accTenPerShare,userCount,tenLPTokenAmount,rewardTenDebt,mineTokenAmount)
        newTenPerBlock = getTenPerBlockByProjectID(_pid);
        (address pairAddr,address tokenAddr, , , , , , ) = tenet.poolSettingInfo(_pid);
        retDatas3 = getPoolPriceInfo(pairAddr,tokenAddr);
//(lpSupply,reserve0,reserve1,pricetype,price0,price1,tokeneth,tokenusdt);
    }

    function getTenPoolBasicInfo() public view returns (address[5] memory retData1,uint256[3] memory retData2,uint256[8] memory retData3,uint256[50] memory retData4,string memory retData5,string memory retData6,string memory retData7) {
        address pairAddr = address(tenet.lpTokenTen());
        address tokenAddr = address(tenet.ten());
        (retData1,retData2,retData5,retData6,retData7) = getPairBasicInfo(pairAddr,tokenAddr);
        (retData3,retData4) = getTenPoolMineInfo();
    }

    function getTokenPoolBasicInfo(uint256 _pid) public view returns (address[5] memory retData1,uint256[3] memory retData2,address[3] memory retData3,uint256[6] memory retData4,string memory retData5,string memory retData6,string memory retData7) {
        (address pairAddr,address tokenAddr, , , , , , ) = tenet.poolSettingInfo(_pid);
        (retData1,retData2,retData5,retData6,retData7) = getPairBasicInfo(pairAddr,tokenAddr);
        (retData3,retData4) = getTokenPoolMineInfo(_pid);
    }

    Function getPoolPriceInfo(Address PairAddr, Address TokenAddr) public view returns (UINT256 [8] Memory Retdatas) {// knownSec //
        address factory = IUniswapV2Pair(pairAddr).factory();
        address token0Addr = IUniswapV2Pair(pairAddr).token0();
        address token1Addr = IUniswapV2Pair(pairAddr).token1();
        retDatas[0] = IUniswapV2Pair(pairAddr).totalSupply();
        (retDatas[1], retDatas[2],) = IUniswapV2Pair(pairAddr).getReserves();
        (retDatas[3],retDatas[4],retDatas[5]) =
calcTokenPrice(IUniswapV2Factory(factory),token0Addr,token1Addr);
        (retDatas[6],retDatas[7]) = calcPrice(uniFactory,tokenAddr);
    }
    function getPairBasicInfo(address pairAddr,address tokenAddr) public view returns (address[5] memory retData1,uint256[3] memory retData2,string memory retData3,string memory retData4,string memory RetData5) {// knownSec // Get basic information about trades
        retData1[0] = IUniswapV2Pair(pairAddr).factory();
        retData1[1] = pairAddr;
        retData1[2] = tokenAddr;
        retData1[3] = IUniswapV2Pair(pairAddr).token0();
        retData1[4] = IUniswapV2Pair(pairAddr).token1();
        (retData2[0],retData3) = getTokenInfo(retData1[2]);
        (retData2[1],retData4) = getTokenInfo(retData1[3]);
        (retData2[2],retData5) = getTokenInfo(retData1[4]);
    }

    function getTenUserPool() public view returns (uint256[6] memory) {
        uint256[6] memory retDatas;
        (retDatas[0],retDatas[1],retDatas[2],retDatas[3]) = tenet.tenUserPool();
        retDatas[4] = tenet.totalAllocPoint();
        retDatas[5] = getTenPerBlockByUser();
        return retDatas;
//(lastRewardBlock,accTenPerShare,allocPoint,lpTokenTotalAmount,totalAllocPoint,newBlockTen);
    }

    function getTenProjectPool() public view returns (uint256[6] memory) {
        uint256[6] memory retDatas;
        (retDatas[0],retDatas[1],retDatas[2],retDatas[3]) = tenet.tenProjectPool();
        retDatas[4] = tenet.totalAllocPoint();
        retDatas[5] = getTenPerBlockByProject();
        return retDatas;
//(lastRewardBlock,accTenPerShare,allocPoint,lpTokenTotalAmount,totalAllocPoint,newBlockTen);
    }

    Function getTenPoolMineInfo() public view returns (UINT256 [8] Memory Retdata1, UINT256 [50] Memory Retdata2) {// knownSec returns () public view returns (UINT256 [8] Memory Retdata2)
        retData1[0] = tenetmine.startBlock();
```

```
        retData1[1] = tenetmine.endBlock();
        retData1[2] = tenetmine.bonusEndBlock();
        retData1[3] = tenetmine.bonus_multiplier();
        retData1[4] = tenetmine.bonusTenPerBlock();
        retData1[5] = tenetmine.subBlockNumerPeriod();
        retData1[6] = tenetmine.totalSupply();
        retData1[7] = tenetmine.getMinePeriodCount();
        for(uint256 i=0; i<tenetmine.getMinePeriodCount(); i++){
            if(i >= 50){
                break;
            }
            (retData2[i], )= tenetmine.allMinePeriodInfo(i);
        }
    }

    Function getTokenPoolmineInfo (UINT256_PID) public view returns (address[3] memory retData1, UINT256 [6] memory retData2) {// knownSec //
        (retData1) = getPoolSettingInfo1(_pid);
        (retData2) = getPoolSettingInfo2(_pid);
    }
    function getPoolSettingInfo1(uint256 _pid) public view returns (address[3] memory retData1) {
        (retData1[0],retData1[1],retData1[2],,,,,,) = tenet.poolSettingInfo(_pid);
    }
    function getPoolSettingInfo2(uint256 _pid) public view returns (uint256[6] memory retData2) {
        (,,,retData2[0],retData2[1],retData2[2],retData2[3],retData2[4],retData2[5]) = tenet.poolSettingInfo(_pid);
    }
    function getPoolInfo(uint256 _pid) public view returns (uint256[8] memory retData3) {
        (retData3[0],retData3[1],retData3[2],retData3[3],retData3[4],retData3[5],retData3[6],retData3[7]) = tenet.poolInfo(_pid);
    }
    function getTokenInfo(address tokenAddr) public view returns (uint256 retData1,string memory retData2) {
        retData1 = ERC20(tokenAddr).decimals();
        retData2 = ERC20(tokenAddr).symbol();
    }

    Function calcPrice(IUNISWAPV2Factory _Factory,address tokenAddr) public view returns (uint256,uint256) {// knownSec //
        uint256 price0 = calcTokenWealth(_factory,tokenAddr,wethAddr);
        uint256 price1 = calcTokenWealth(_factory,tokenAddr,wusdtAddr);
        return (price0,price1);
    }

    Function calcTokenPrice(IUNISWAPV2Factory _Factory,address token0Addr,address token1Addr) public view returns (uint256,uint256,uint256)
        uint256 pricetype = 0;
        uint256 price0 = 0;
        uint256 price1 = 0;
        price0 = calcTokenWealth(_factory,token0Addr,wethAddr);
        if(price0 == 0){
            price1 = calcTokenWealth(_factory,token1Addr,wethAddr);
            if(price1 == 0){
                pricetype = 1;
                price0 = calcTokenWealth(_factory,token0Addr,wusdtAddr);
                if(price0 == 0){
                    price1 = calcTokenWealth(_factory,token1Addr,wusdtAddr);
                }
            }
        }
        return (pricetype,price0,price1);
    }

    function calcETHPrice() public view returns (uint256) {
        IUniswapV2Pair pair = IUniswapV2Pair(uniFactory.getPair(wethAddr,wusdtAddr));
        if (address(pair) == address(0)) {
            return 0;
        }
        address token0 = pair.token0();
        (uint reserve0, uint reserve1,) = pair.getReserves();
        if(token0 == wethAddr){
            return reserve1.mul(MINWEALTH_AMOUNT).div(reserve0);
        }
        return reserve0.mul(MINWEALTH_AMOUNT).div(reserve1);
    }

    function calcTokenWealth(IUniswapV2Factory _factory,address token,address wealth) public view returns (uint256) {
        if (token == wealth) {
            return MINWEALTH_AMOUNT;
        }
        IUniswapV2Pair pair = IUniswapV2Pair(_factory.getPair(token, wealth));
        if (address(pair) == address(0)) {
            return 0;
        }
        (uint reserve0, uint reserve1,) = pair.getReserves();
        if(token == pair.token0()){
            return reserve1.mul(MINWEALTH_AMOUNT).div(reserve0);
```

```
            }
            return reserve0.mul(MINWEALTH_AMOUNT).div(reserve1);
        }

        function getTenPerBlockByUser() public view returns (uint256 tenReward) {
            uint256[2] memory allTmpData;      //allocPoint,lpSupply
            (,,allTmpData[0],allTmpData[1]) = tenet.tenUserPool();
            if (allTmpData[1] < MINLPTOKEN_AMOUNT) {
                return 0;
            }
            tenReward = tenetmine.calcMineTenReward(block.number-1, block.number);
            tenReward = tenReward.mul(allTmpData[0]).div(tenet.totalAllocPoint());
        }

        function getTenPerBlockByProject() public view returns (uint256 tenReward) {
            uint256[3] memory allTmpData;      //lpTokenAmount,allocPoint,tenLPTokenAmount
            ( , ,allTmpData[0],allTmpData[1]) = tenet.tenProjectPool();
            if (allTmpData[1] < MINLPTOKEN_AMOUNT) {
                return 0;
            }
            tenReward = tenetmine.calcMineTenReward(block.number-1, block.number);
            tenReward = tenReward.mul(allTmpData[0]).div(tenet.totalAllocPoint());
        }

        function getTenPerBlockByProjectID(uint _pid) public view returns (uint256 tenReward) {
            uint256[4] memory allTmpData;      //lpTokenAmount,allocPoint,tenLPTokenAmount
            ( ,allTmpData[0], , , ,allTmpData[3], , ) = tenet.poolInfo(_pid);
            if(allTmpData[0] < MINLPTOKEN_AMOUNT){
                return 0;
            }
            if (allTmpData[3] < MINLPTOKEN_AMOUNT) {
                return 0;
            }
            ( , ,allTmpData[1],allTmpData[2]) = tenet.tenProjectPool();
            tenReward = tenetmine.calcMineTenReward(block.number-1, block.number);
            tenReward                                                                                    =
tenReward.mul(allTmpData[3]).mul(allTmpData[1]).div(tenet.totalAllocPoint()).div(allTmpData[2]);
        }
}
```

TenetSwap.sol

```
// SPDX-License-Identifier: MIT

Pragma solidity 0.6.12;


import "./openzeppelin/contracts/token/ERC20/IERC20.sol";
import "./openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "./openzeppelin/contracts/utils/EnumerableSet.sol";
import "./openzeppelin/contracts/math/SafeMath.sol";
import "./openzeppelin/contracts/access/Ownable.sol";
import "./uniswapv2/IUniswapV2Pair.sol";
import "./uniswapv2/IUniswapV2Factory.sol";
import "./Tenet.sol";
interface IWETH {
    function deposit() external payable;
    function transfer(address to, uint value) external returns (bool);
    function withdraw(uint) external;
}
contract TenetSwap is Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    Tenet public tenetAddr;
    address public wethAddr;

    modifier ensure(uint deadline) {
        require(deadline >= block.timestamp, 'TenetSwap: EXPIRED');
        _;
    }
    event TransferTokenToLPToken(address indexed user, uint256 indexed pid, address tokenAddr,uint256
tokenAmount,address lpTokenAddr, uint256 lpTenAmount);
    event TransferTokensToLPToken(address indexed user, uint256 indexed pid, uint256 token0Amount,uint256
token1Amount,address lpTokenAddr, uint256 lpTenAmount);
    event ChangeLPToken(address indexed user,address lpTokenAddr,uint256 token0Amount,uint256
token1Amount,uint256 lpTenAmount);

    constructor(address _tenetAddr,address _wethAddr) public {
        tenetAddr = Tenet(_tenetAddr);
        wethAddr = _wethAddr;
    }
    Function set_tenet(Tenet _tenet) public onlyOwner {/// knownSec
        tenetAddr = _tenet;
    }
    receive() external payable {
        assert(msg.sender == wethAddr);      // only accept ETH via fallback from the WETH contract
```

```
    }
    function _calcLiquidAmountIn(address _pairAddr,uint256[2] memory _amountDesired,uint256 _amountMinRate) internal virtual view returns (uint amount0, uint amount1) {
        require(_amountMinRate <= 1000, 'addLiquidity: INSUFFICIENT_AMOUNT_MINRATE');
        uint256[2] memory _amountMin;
        _amountMin[0] = _amountDesired[0].mul(_amountMinRate).div(1000);
        _amountMin[1] = _amountDesired[1].mul(_amountMinRate).div(1000);
        uint256[2] memory _reserve;
        (_reserve[0], _reserve[1],) = IUniswapV2Pair(_pairAddr).getReserves();
        uint amount1Optimal = _amountDesired[0].mul(_reserve[1]).div(_reserve[0]);
        if (amount1Optimal <= _amountDesired[1]) {
            require(amount1Optimal >= _amountMin[1], '_addLiquidity: INSUFFICIENT_1_AMOUNT');
            (amount0, amount1) = (_amountDesired[0], amount1Optimal);
        } else {
            uint amount0Optimal = _amountDesired[1].mul(_reserve[0]).div(_reserve[1]);
            assert(amount0Optimal <= _amountDesired[0]);
            require(amount0Optimal >= _amountMin[0], '_addLiquidity: INSUFFICIENT_0_AMOUNT');
            (amount0, amount1) = (amount0Optimal, _amountDesired[1]);
        }
    }
    function _transferToPair(address _tokenAddr,address _fromAddr, address _toAddr, uint256 _value) internal {
        if(_tokenAddr == wethAddr){
            IWETH(_tokenAddr).transfer(_toAddr, _value);
        }else{
            if(_fromAddr == address(this)){
                IERC20(_tokenAddr).transfer(_toAddr, _value);
            }else{
                IERC20(_tokenAddr).transferFrom(_fromAddr,_toAddr, _value);
            }
        }
    }
    function _returnToUser(address _tokenAddr,address _fromAddr,uint256 _value) internal{
        if(_value > 0){
            if(_tokenAddr == wethAddr){
                IWETH(_tokenAddr).withdraw(_value);
                msg.sender.transfer(_value);
            }else{
                if(_fromAddr == address(this)){
                    IERC20(_tokenAddr).transfer(msg.sender, _value);
                }
            }
        }
    }
    Function _addLiquidity(address _fromAddr,address _pairAddr,uint256[2] memory _amountDesired,address to,uint256 _amountMinRate) internal returns (uint256) {/// knownSec //
        address[2] memory _tokenAddr;
        _tokenAddr[0] = IUniswapV2Pair(_pairAddr).token0();
        _tokenAddr[1] = IUniswapV2Pair(_pairAddr).token1();
        (uint256 amountA,uint256 amountB) = _calcLiquidAmountIn(_pairAddr, _amountDesired, _amountMinRate);
        _transferToPair(_tokenAddr[0], _fromAddr, _pairAddr,amountA);
        _transferToPair(_tokenAddr[1], _fromAddr, _pairAddr,amountB);
        uint256 liquidity = IUniswapV2Pair(_pairAddr).mint(to);
        _returnToUser(_tokenAddr[0], _fromAddr, _amountDesired[0].sub(amountA));
        _returnToUser(_tokenAddr[1], _fromAddr, _amountDesired[1].sub(amountB));
        return liquidity;
    }
    Function getPrice(address _pairAddr, address _fromAddr) public view returns (uint256) {/// knownSec //
        (uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(_pairAddr).getReserves();
        if(_fromAddr == IUniswapV2Pair(_pairAddr).token0()){
            return reserve1.mul(1e12).div(reserve0);
        }else{
            return reserve0.mul(1e12).div(reserve1);
        }
    }
    Function getAmountOut(Address _PairAddr, Address _FromAddr, Uint Amountin) public view virtual returns (UINT256)
        require(amountIn > 0, 'getAmountOut: INSUFFICIENT_INPUT_AMOUNT');
        (uint256 reserve0, uint256 reserve1,) = IUniswapV2Pair(_pairAddr).getReserves();
        require(reserve0 > 0 && reserve1 > 0, 'getAmountOut: INSUFFICIENT_LIQUIDITY');
        if(_fromAddr == IUniswapV2Pair(_pairAddr).token0()){
            uint amountInWithFee = amountIn.mul(997);
            uint numerator = amountInWithFee.mul(reserve1);
            uint denominator = reserve0.mul(1000).add(amountInWithFee);
            return numerator.div(denominator);
        }else{
            uint amountInWithFee = amountIn.mul(997);
            uint numerator = amountInWithFee.mul(reserve0);
            uint denominator = reserve1.mul(1000).add(amountInWithFee);
            return numerator.div(denominator);
        }
    }
    Function _swapToken(address _pairAddr, address _fromAddr,uint256 _tokenAmount) internal returns (uint256) {///knownsec//
        uint256 tokenAmountOut = getAmountOut(_pairAddr,_fromAddr,_tokenAmount);
        if(_fromAddr == wethAddr){
```

```
                IWETH(_fromAddr).transfer(_pairAddr, _tokenAmount);
            }else{
                IERC20(_fromAddr).transfer(_pairAddr, _tokenAmount);
            }
            if(_fromAddr == IUniswapV2Pair(_pairAddr).token0()){
                IUniswapV2Pair(_pairAddr).swap(0, tokenAmountOut, address(this), new bytes(0));
            }else{
                IUniswapV2Pair(_pairAddr).swap(tokenAmountOut, 0, address(this), new bytes(0));
            }
            return tokenAmountOut;
        }
        function _transferTokensOut(address _pairAddr,address _tokenAddr,uint256 _tokenAmount) internal returns
(uint256,uint256) {
            IUniswapV2Factory factory = IUniswapV2Factory(IUniswapV2Pair(_pairAddr).factory());
            require(address(factory) != address(0), 'transferTokensOut: INSUFFICIENT_PAIR');
            uint256[8] memory dataAll;
            (dataAll[6], dataAll[7],) = IUniswapV2Pair(_pairAddr).getReserves();
            require(dataAll[6] > 0, 'transferTokensOut: INSUFFICIENT_RESERVE0');
            require(dataAll[7] > 0, 'transferTokensOut: INSUFFICIENT_RESERVE1');
            address[2] memory allPairAddr;
            allPairAddr[0] = factory.getPair(_tokenAddr,IUniswapV2Pair(_pairAddr).token0());
            require(allPairAddr[0] != address(0), 'transferTokensOut: INVALID_PAIR0');
            dataAll[0] = getPrice(allPairAddr[0],_tokenAddr);
            allPairAddr[1] = factory.getPair(_tokenAddr,IUniswapV2Pair(_pairAddr).token1());
            require(allPairAddr[1] != address(0), 'transferTokensOut: INVALID_PAIR1');
            dataAll[1] = getPrice(allPairAddr[1],_tokenAddr);
            dataAll[2]                                                                                    =
_tokenAmount.mul(dataAll[1]).mul(dataAll[6]).div(dataAll[0].mul(dataAll[7]).add(dataAll[1].mul(dataAll[6])));
            dataAll[3] = _tokenAmount.sub(dataAll[2]);
            dataAll[4] = _swapToken(allPairAddr[0],_tokenAddr,dataAll[2]);
            dataAll[5] = _swapToken(allPairAddr[1],_tokenAddr,dataAll[3]);
            return (dataAll[4],dataAll[5]);
        }
        function _transferTokenXOut(address _pairAddr,address _tokenAddr,uint256 _tokenAmount,uint256
tokenType) internal returns (uint256,uint256) {
            IUniswapV2Factory factory = IUniswapV2Factory(IUniswapV2Pair(_pairAddr).factory());
            require(address(factory) != address(0), 'transferTokenXOut: INSUFFICIENT_PAIR');
            uint256[4] memory dataAll;
            (dataAll[0], dataAll[1],) = IUniswapV2Pair(_pairAddr).getReserves();
            require(dataAll[0] > 0, 'transferTokenXOut: INSUFFICIENT_RESERVE0');
            require(dataAll[1] > 0, 'transferTokenXOut: INSUFFICIENT_RESERVE1');
            dataAll[2] = _tokenAmount.div(2);
            dataAll[3] = _swapToken(_pairAddr,_tokenAddr,dataAll[2]);
            if(tokenType == 0){
                return (dataAll[2],dataAll[3]);
            }else{
                return (dataAll[3],dataAll[2]);
            }
        }
        function _transferLPToken(uint256 _poolType,uint256 _pid,address _pairAddr,uint256[2] memory
_tokenAmountOut,uint256 _amountMinRate) internal returns (uint256) {
            uint                                                                                    liquidity
=
_addLiquidity(address(this),_pairAddr,_tokenAmountOut,address(this),_amountMinRate);
            IERC20(_pairAddr).approve(address(tenetAddr),liquidity);
            if(_poolType == 0){
                tenetAddr.depositTenByUserFrom(msg.sender,liquidity);
            }else{
                tenetAddr.depositLPTokenFrom(msg.sender,_pid,liquidity);
            }
            return liquidity;
        }
        function transferTokenToLPToken(uint256 _poolType,uint256 _pid,address _pairAddr,address
_tokenAddr,uint256 _tokenAmount,uint256 _amountMinRate,uint256 deadline) public virtual ensure(deadline) {
            if(_tokenAddr != wethAddr){
                IERC20(_tokenAddr).transferFrom(msg.sender, address(this),_tokenAmount);
            }
            uint256[2] memory tokenAmountOut;
            if(_tokenAddr == IUniswapV2Pair(_pairAddr).token0()){
                (tokenAmountOut[0],tokenAmountOut[1])                                                =
_transferTokenXOut(_pairAddr,_tokenAddr,_tokenAmount,0);
            }else if(_tokenAddr == IUniswapV2Pair(_pairAddr).token1()){
                (tokenAmountOut[0],tokenAmountOut[1])                                                =
_transferTokenXOut(_pairAddr,_tokenAddr,_tokenAmount,1);
            }else{
                (tokenAmountOut[0],tokenAmountOut[1])                                                =
_transferTokensOut(_pairAddr,_tokenAddr,_tokenAmount);
            }
            uint liquidity = _transferLPToken(_poolType,_pid,_pairAddr,tokenAmountOut,_amountMinRate);
            emit TransferTokenToLPToken(msg.sender,_pid,_tokenAddr,_tokenAmount,_pairAddr,liquidity);
        }
        function transferETHToLPToken(uint256 _poolType,uint256 _pid,address _pairAddr,uint256
_amountMinRate,uint256 deadline) external virtual payable ensure(deadline) {
            IWETH(wethAddr).deposit{value: msg.value}();
            transferTokenToLPToken(_poolType,_pid,_pairAddr,wethAddr,msg.value,_amountMinRate,deadline);
        }
        function transferTokensToLPToken(uint256 _poolType,uint256 _pid,address _pairAddr,uint256
_token0Amount,uint256 _token1Amount,uint256 _amountMinRate,uint256 deadline) public virtual
```

```
ensure(deadline) {
        uint256[2] memory tokenAmountOut;
        tokenAmountOut[0] = _token0Amount;
        tokenAmountOut[1] = _token1Amount;
        if(wethAddr ! = IUniswapV2Pair(_pairAddr).token0()){
            IERC20(IUniswapV2Pair(_pairAddr).token0()).transferFrom(msg.sender, address(this), tokenAmountOut[0]);
        }
        if(wethAddr ! = IUniswapV2Pair(_pairAddr).token1()){
            IERC20(IUniswapV2Pair(_pairAddr).token1()).transferFrom(msg.sender, address(this), tokenAmountOut[1]);
        }
        uint liquidity = _transferLPToken(_poolType, _pid, _pairAddr,tokenAmountOut,_amountMinRate);
        emit TransferTokensToLPToken(msg.sender,_pid,tokenAmountOut[0],tokenAmountOut[1],_pairAddr,liquidity);
    }
    function transferETHsToLPToken(uint256 _poolType,uint256 _pid,address _pairAddr,uint256 _tokenAmount,uint256 _amountMinRate,uint256 deadline) external virtual payable ensure(deadline) {
        IWETH(wethAddr).deposit{value: msg.value}();
        if(wethAddr == IUniswapV2Pair(_pairAddr).token0()){
            transferTokensToLPToken(_poolType, _pid, _pairAddr,msg.value, _tokenAmount, _amountMinRate,deadline);
        }else{
            transferTokensToLPToken(_poolType, _pid, _pairAddr, _tokenAmount,msg.value, _amountMinRate,deadline);
        }
    }
    function changeLPToken(address _pairAddr,uint256[2] memory _tokenAmountOut,uint256 _amountMinRate,uint256 deadline) public ensure(deadline){
        uint liquidity = _addLiquidity(msg.sender, _pairAddr, _tokenAmountOut,msg.sender, _amountMinRate);
        emit ChangeLPToken(msg.sender, _pairAddr, _tokenAmountOut[0], _tokenAmountOut[1],liquidity);
    }
    function changeWethLPToken(address _pairAddr,uint256 _tokenAmount,uint256 _amountMinRate,uint256 deadline) external payable ensure(deadline){
        uint256[2] memory tokenAmountOut;
        if(wethAddr == IUniswapV2Pair(_pairAddr).token0()){
            tokenAmountOut[0] = msg.value;
            tokenAmountOut[1] = _tokenAmount;
        }else{
            tokenAmountOut[0] = _tokenAmount;
            tokenAmountOut[1] = msg.value;
        }
        IWETH(wethAddr).deposit{value: msg.value}();
        changeLPToken(_pairAddr,tokenAmountOut, _amountMinRate,deadline);
    }
}
```

TenetToken.sol

```
// SPDX-License-Identifier: MIT

Pragma solidity 0.6.12;


import "./openzeppelin/contracts/token/ERC20/ERC20.sol";
import "./openzeppelin/contracts/access/Ownable.sol";


// TntToken with Governance.
contract TenetToken is ERC20("Tenet", "TEN"), Ownable {
    // @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
    function mint(address _to, uint256 _amount) public onlyOwner {
        _mint(_to, _amount);
        _moveDelegates(address(0), _delegates[_to], _amount);
    }
    function burn(address _account, uint256 _amount) public onlyOwner {
        _burn(_account, _amount);
    }
    // Copied and modified from YAM code:
    // https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAMGovernanceStorage.sol
    // https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAMGovernance.sol
    // Which is copied and modified from COMPOUND:
    // https://github.com/compound-finance/compound-protocol/blob/master/contracts/Governance/Comp.sol

    // @notice A record of each accounts delegate
    mapping (address => address) internal _delegates;

    // @notice A checkpoint for marking number of votes from a given block
    struct Checkpoint {
        uint32 fromBlock;
        uint256 votes;
    }

    // @notice A record of votes checkpoints for each account, by index
    mapping (address => mapping (uint32 => Checkpoint)) public checkpoints;
```

```
        // @notice The number of checkpoints for each account
        mapping (address => uint32) public numCheckpoints;

        // @notice The EIP-712 typehash for the contract's domain
        bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");

        // @notice The EIP-712 typehash for the delegation struct used by the contract
        bytes32 public constant DELEGATION_TYPEHASH = keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)");

        // @notice A record of states for signing / validating signatures
        mapping (address => uint) public nonces;

        // @notice An event thats emitted when an account changes its delegate
        event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);

        // @notice An event thats emitted when a delegate account's vote balance changes
        event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);

        /**
         * @notice Delegate votes from `msg.sender` to `delegatee`
         * @param delegator The address to get delegatee for
         */
        function delegates(address delegator)
            external
            view
            returns (address)
        {
            return _delegates[delegator];
        }

       /**
        * @notice Delegate votes from `msg.sender` to `delegatee`
        * @param delegatee The address to delegate votes to
        */
        function delegate(address delegatee) external {
            return _delegate(msg.sender, delegatee);
        }

        /**
         * @notice Delegates votes from signatory to `delegatee`
         * @param delegatee The address to delegate votes to
         * @param nonce The contract state required to match the signature
         * @param expiry The time at which to expire the signature
         * @param v The recovery byte of the signature
         * @param r Half of the ECDSA signature pair
         * @param s Half of the ECDSA signature pair
         */
        Function DelegateBySig (/// knownSec
            address delegatee,
            uint nonce,
            uint expiry,
            uint8 v,
            bytes32 r,
            bytes32 s
        )
            external
        {
            bytes32 domainSeparator = keccak256(
                abi.encode(
                    DOMAIN_TYPEHASH,
                    keccak256(bytes(name())),
                    getChainId(),
                    address(this)
                )
            );

            bytes32 structHash = keccak256(
                abi.encode(
                    DELEGATION_TYPEHASH,
                    delegatee,
                    nonce,
                    expiry
                )
            );

            bytes32 digest = keccak256(
                abi.encodePacked(
                    "\x19\x01",
                    domainSeparator,
                    structHash
                )
            );

            address signatory = ecrecover(digest, v, r, s);
```

```
        require(signatory != address(0), "delegateBySig: invalid signature");
        require(nonce == nonces[signatory]++, "delegateBySig: invalid nonce");
        require(now <= expiry, "delegateBySig: signature expired");
        return _delegate(signatory, delegatee);
    }

    /**
     * @notice Gets the current votes balance for `account`
     * @param account The address to get votes balance
     * @return The number of current votes for `account`
     */
    function getCurrentVotes(address account)
        external
        view
        returns (uint256)
    {// knownSec // Get the current vote
        uint32 nCheckpoints = numCheckpoints[account];
        return nCheckpoints > 0 ?      checkpoints[account][nCheckpoints - 1].votes : 0;
    }

    /**
     * @notice Determine the prior number of votes for an account as of a block number
     * @dev Block number must be a finalized block or else this function will revert to prevent misinformation.
     * @param account The address of the account to check
     * @param blockNumber The block number to get the vote balance at
     * @return The number of votes the account had as of the given block
     */
    function getPriorVotes(address account, uint blockNumber)
        external
        view
        returns (uint256)
    {// knownSec // Get the previous vote
        require(blockNumber < block.number, "getPriorVotes: not yet determined");

        uint32 nCheckpoints = numCheckpoints[account];
        if (nCheckpoints == 0) {
            return 0;
        }

        // First check most recent balance
        if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
            return checkpoints[account][nCheckpoints - 1].votes;
        }

        // Next check implicit zero balance
        if (checkpoints[account][0].fromBlock > blockNumber) {
            return 0;
        }

        uint32 lower = 0;
        uint32 upper = nCheckpoints - 1;
        while (upper > lower) {
            uint32 center = upper - (upper - lower) / 2;      // ceil, avoiding overflow
            Checkpoint memory cp = checkpoints[account][center];
            if (cp.fromBlock == blockNumber) {
                return cp.votes;
            } else if (cp.fromBlock < blockNumber) {
                lower = center;
            } else {
                upper = center - 1;
            }
        }
        return checkpoints[account][lower].votes;
    }

    function _delegate(address delegator, address delegatee)
        internal
    {
        address currentDelegate = _delegates[delegator];
        uint256 delegatorBalance = balanceOf(delegator);      // balance of underlying TNTs (not scaled);
        _delegates[delegator] = delegatee;

        emit DelegateChanged(delegator, currentDelegate, delegatee);

        _moveDelegates(currentDelegate, delegatee, delegatorBalance);
    }

    function _moveDelegates(address srcRep, address dstRep, uint256 amount) internal {
        if (srcRep != dstRep && amount > 0) {
            if (srcRep != address(0)) {
                // decrease old representative
                uint32 srcRepNum = numCheckpoints[srcRep];
                uint256 srcRepOld = srcRepNum > 0 ?      checkpoints[srcRep][srcRepNum - 1].votes : 0;
                uint256 srcRepNew = srcRepOld.sub(amount);
                _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
            }
```

```
                if (dstRep != address(0)) {
                    // increase new representative
                    uint32 dstRepNum = numCheckpoints[dstRep];
                    uint256 dstRepOld = dstRepNum > 0 ?      checkpoints[dstRep][dstRepNum - 1].votes : 0;
                    uint256 dstRepNew = dstRepOld.add(amount);
                    _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
                }
            }
        }

        function _writeCheckpoint(
            address delegatee,
            uint32 nCheckpoints,
            uint256 oldVotes,
            uint256 newVotes
        )
            internal
        {
            uint32 blockNumber = safe32(block.number, "_writeCheckpoint: block number exceeds 32 bits");

            if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
                checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
            } else {
                checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
                numCheckpoints[delegatee] = nCheckpoints + 1;
            }

            emit DelegateVotesChanged(delegatee, oldVotes, newVotes);
        }

        function safe32(uint n, string memory errorMessage) internal pure returns (uint32) {
            require(n < 2**32, errorMessage);
            return uint32(n);
        }

        Function getChainId() internal pure returns (uint) {// knownSec //
            uint256 chainId;
            assembly { chainId := chainid() }
            return chainId;
        }
}
```

Timelock.sol

```
// SPDX-License-Identifier: MIT
//                                    COPIED                                    FROM
https://github.com/compound-finance/compound-protocol/blob/master/contracts/Governance/GovernorAlpha.sol
// Copyright 2020 Compound Labs, Inc.
// Redistribution and use in source and binary forms, with or without modification, are permitted provided that the
following conditions are met:
// 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following
disclaimer.
// 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the
following disclaimer in the documentation and/or other materials provided with the distribution.
// 3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote
products derived from this software without specific prior written permission.
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY
EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;      LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
DAMAGE.
//
// Ctrl+f for XXX to see all the modifications.

// XXX: pragma solidity ^0.5.16;
Pragma solidity 0.6.12;

// XXX: import "./SafeMath.sol";
import "./openzeppelin/contracts/math/SafeMath.sol";

contract Timelock {
    using SafeMath for uint;

    event NewAdmin(address indexed newAdmin);
    event NewPendingAdmin(address indexed newPendingAdmin);
    event NewDelay(uint indexed newDelay);
    event CancelTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature,
bytes data, uint eta);
    event ExecuteTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature,
bytes data, uint eta);
    event QueueTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes
```

```
data, uint eta);

    uint public constant GRACE_PERIOD = 14 days;
    uint public constant MINIMUM_DELAY = 2 days;
    uint public constant MAXIMUM_DELAY = 30 days;

    address public admin;
    address public pendingAdmin;
    uint public delay;
    bool public admin_initialized;

    mapping (bytes32 => bool) public queuedTransactions;


    constructor(address admin_, uint delay_) public {
        require(delay_ >= MINIMUM_DELAY, "Timelock::constructor: Delay must exceed minimum delay.");
        require(delay_ <= MAXIMUM_DELAY, "Timelock::constructor: Delay must not exceed maximum
delay.");

        admin = admin_;
        delay = delay_;
        admin_initialized = false;
    }

    // XXX: function() external payable { }
    receive() external payable { }

    function setDelay(uint delay_) public {
        require(msg.sender == address(this), "Timelock::setDelay: Call must come from Timelock.");
        require(delay_ >= MINIMUM_DELAY, "Timelock::setDelay: Delay must exceed minimum delay.");
        require(delay_ <= MAXIMUM_DELAY, "Timelock::setDelay: Delay must not exceed maximum
delay.");
        delay = delay_;

        emit NewDelay(delay);
    }

    function acceptAdmin() public {
        require(msg.sender == pendingAdmin, "Timelock::acceptAdmin: Call must come from
pendingAdmin.");
        admin = msg.sender;
        pendingAdmin = address(0);

        emit NewAdmin(admin);
    }

    function setPendingAdmin(address pendingAdmin_) public {
        // allows one time setting of admin for deployment purposes
        if (admin_initialized) {
            require(msg.sender == address(this), "Timelock::setPendingAdmin: Call must come from
Timelock.");
        } else {
            require(msg.sender == admin, "Timelock::setPendingAdmin: First call must come from admin.");
            admin_initialized = true;
        }
        pendingAdmin = pendingAdmin_;

        emit NewPendingAdmin(pendingAdmin);
    }

    function queueTransaction(address target, uint value, string memory signature, bytes memory data, uint eta)
public returns (bytes32) {
        require(msg.sender == admin, "Timelock::queueTransaction: Call must come from admin.");
        require(eta >= getBlockTimestamp().add(delay), "Timelock::queueTransaction: Estimated execution
block must satisfy delay.");

        bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
        queuedTransactions[txHash] = true;

        emit QueueTransaction(txHash, target, value, signature, data, eta);
        return txHash;
    }

    function cancelTransaction(address target, uint value, string memory signature, bytes memory data, uint eta)
public {
        require(msg.sender == admin, "Timelock::cancelTransaction: Call must come from admin.");

        bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
        queuedTransactions[txHash] = false;

        emit CancelTransaction(txHash, target, value, signature, data, eta);
    }

    function executeTransaction(address target, uint value, string memory signature, bytes memory data, uint
eta) public payable returns (bytes memory) {
        require(msg.sender == admin, "Timelock::executeTransaction: Call must come from admin.");
```

```
            bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
            require(queuedTransactions[txHash], "Timelock::executeTransaction: Transaction hasn't been queued.");
            require(getBlockTimestamp() >= eta, "Timelock::executeTransaction: Transaction hasn't surpassed time lock.");
            require(getBlockTimestamp() <= eta.add(GRACE_PERIOD), "Timelock::executeTransaction: Transaction is stale.");

            queuedTransactions[txHash] = false;

            bytes memory callData;

            if (bytes(signature).length == 0) {
                callData = data;
            } else {
                callData = abi.encodePacked(bytes4(keccak256(bytes(signature))), data);
            }

            // solium-disable-next-line security/no-call-value
            (bool success, bytes memory returnData) = target.call.value(value)(callData);
            require(success, "Timelock::executeTransaction: Transaction execution reverted.");

            emit ExecuteTransaction(txHash, target, value, signature, data, eta);

            return returnData;
        }

        function getBlockTimestamp() internal view returns (uint) {
            // solium-disable-next-line security/no-block-members
            return block.timestamp;
        }
}
```

6. Appendix B: Safety risk rating criteria

Smart contract vulnerability rating criteria	
Vulnerability rating	**Vulnerability rating description**
High risk vulnerabilities	Vulnerability that can directly cause the loss of the token contract or the user's capital, such as: numerical overflow vulnerability that can cause the value of the token to zero, false recharge vulnerability that can cause the loss of the exchange token, reentrant vulnerability that can cause the loss of the contract account ETH or the token, etc. Vulnerabilities that can cause the loss of the ownership of the token contract, such as access control defects of key functions, access control bypass of key functions caused by Call injection, etc. Vulnerability that can cause a token contract to not work properly, such as Denial of Service (DoS) due to sending ETH to a malicious address, DoS due to running out of gas.
In the dangerous holes	High-risk vulnerabilities that require a specific address to trigger, such as numerical overflow vulnerabilities that can only be triggered by the owner of a token contract; Access control defects of non-critical functions, logic design defects that cannot cause direct loss of funds, etc.
Low latent loophole	Vulnerability that is difficult to trigger, vulnerability that does limited harm after triggering, such as numerical overflow vulnerability that requires a large number of ETH or tokens to trigger, vulnerability that the attacker cannot directly profit after triggering numerical overflow, transaction sequence dependency risk triggered by specifying high GAS, etc.

7. Appendix C: Introduction to Smart Contract Security Audit Tools

6.1 Manticore

Manticore is a symbolic execution tool for analyzing binaries and smart contracts. Manticore includes a symbolic Ethereum virtual machine (EVM), an EVM disassembler/assembler, and a handy interface for automatically compiling and analyzing Solidity. It also integrates Ethersplay, a visualized disassembler of Bit of Traits of Bits for EVM bytecode for visual analysis. Like binaries, Manticore provides a simple command-line interface and a Python API for analyzing EVM bytecode.

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 ordering dependencies, etc. Conveniently, Oyente's design is modular, so this allows advanced users to implement and insert their own instrumentation logic to check custom properties in their contracts.

6.3 securify. Sh

Securify can verify common security issues with Ethereum smart contracts, such as out-of-order transactions and lack of input validation. It analyzes all possible execution paths of programs in a fully automated manner. In addition, Securify has a specific language for specifying vulnerabilities, which allows Securify to keep an eye on current security and other reliability issues.

6.4 Echidna

Echidna is a Haskell library designed for fuzzy testing of EVM code.

6.5 MAIAN

MAIAN is an automated tool used to find vulnerabilities in Ethereum smart contracts. MAIAN processes the bytecode of contracts and tries to build a series of transactions to find and confirm errors.

6.6 ethersplay

Ethersplay is an EVM disassembler that includes correlation analysis tools.

6.7 IDA - evm entry

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

6.8 want - ide

Remix is a browser-based compiler and IDE that allows users to build Ethereum contracts and debug transactions using the Solidity language.

6.9 Know Chuang Yu Blockchain Security Auditor Special Toolkit

Know-how Penetration Tester Kit is developed, collected and used by Know-how Penetration Tester Engineers, which includes batch automated test tools, self-developed tools, scripts or utilization tools, etc.

知道创宇

北京知道创宇信息技术股份有限公司

咨询电话 +86(10)400 060 9587

邮　　箱 sec@knownsec.com

官　　网 www.knownsec.com

地　　址 北京市 朝阳区 望京 SOHO T2-B座-2509