Competitive Security Assessment

Quantemo

Jun 8th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	Quantemo
Platform & Language	Solidity
Codebase	• Code shared by zip file • audit zip file MD5 - eaaf7aaf7140ed98e4e74d29463ba61c • final zip file MD5 - c8afabe35edd20eef6cdf78af66be803
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

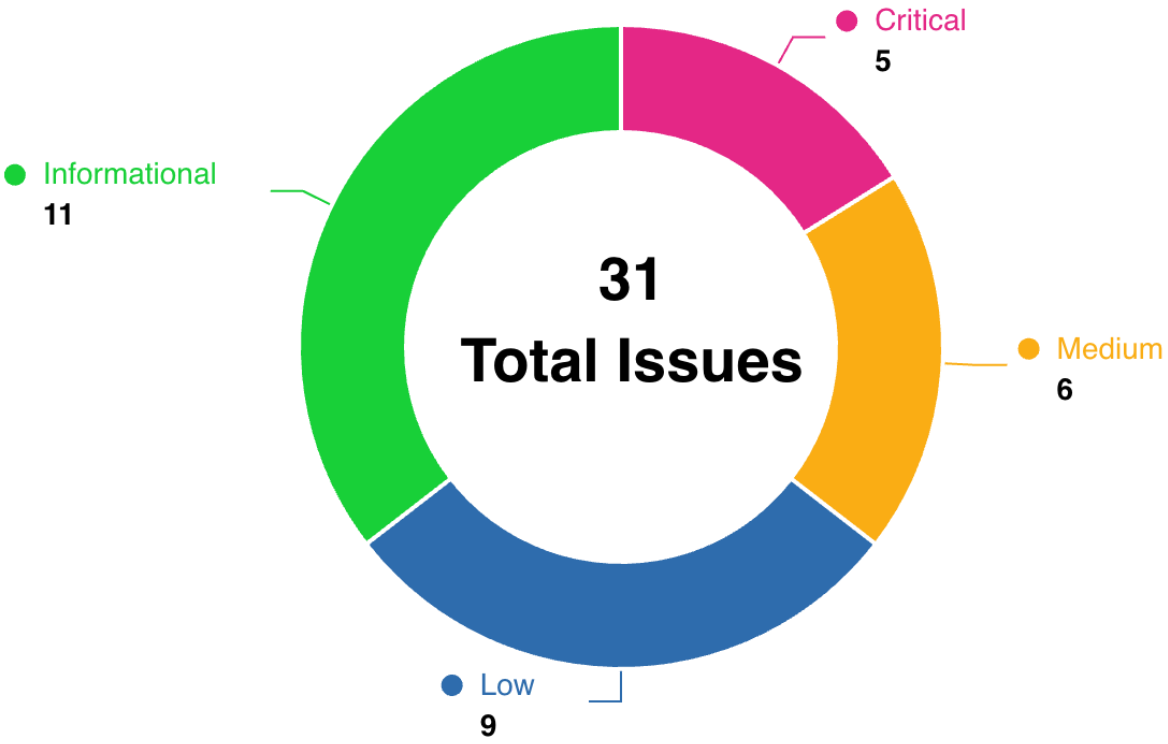
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	5	0	0	3	0	2
Medium	6	0	0	6	0	0
Low	9	0	1	6	0	2
Informational	11	0	1	9	0	1

Audit Scope

File	MD5
./my_factory.sol	feef72bf025ba9575a29e558ec83f6d1
./dynamic.sol	d34dd2d268b362b6483cd97a2074ea6c
./router.sol	86c3df52cc3508d44c8f4947a7be979c
./QEMO.sol	3198ed472972c32e81a2a7de06318899
./stake.sol	d4dfbcf07788260dabe7592b23425ea6

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
QTM-1	Incorrect unit in `QEMOStake`::`buyID0()`	Language Specific	Critical	Declined	Hupixiong3
QTM-2	Integer Overflow	DOS	Critical	Fixed	infinityhacker
QTM-3	Logical error:Use `referReward` instead of `reward`	Logical	Critical	Fixed	biakia
QTM-4	Missing K check in swap function	Logical	Critical	Declined	infinityhacker, biakia, Xi_Zi

QTM-5	Vulnerability in `skim` Function of `my_factory.sol` Contract	Flash Loan Attacks	Critical	Fixed	Xi_Zi
QTM-6	Centralized risk.	Logical	Medium	Fixed	Hupixiong3
QTM-7	Logical error:IDO funds will be permanently locked in the contract	Logical	Medium	Fixed	biakia
QTM-8	Logical error:Missing check IDO Time Range in `stake:: setIDOTime`	Logical	Medium	Fixed	Xi_Zi
QTM-9	Logical error:Missing check startTime in `stake:: countingStakePower`	Logical	Medium	Fixed	Xi_Zi
QTM-10	Logical error:Referral rewards will be distributed twice in function `IdoStake`	Logical	Medium	Fixed	biakia
QTM-11	Logical error:Some usdt may be permanently locked after swapping in the contract `QEMOStake`	Logical	Medium	Fixed	biakia
QTM-12	Invalid Balance Checks	Logical	Low	Fixed	biakia
QTM-13	Logical error:Missing update `rate` after `dailyOut` changed	Logical	Low	Declined	biakia
QTM-14	Logical error:Missing update `refer_R` of the invitor when calling function `stakeFor`	Logical	Low	Declined	biakia
QTM-15	Missing 0 address check in `QEMOFactory`::`setFeeToSetter()` and `CnnRouter`::`changeOwner()`	Logical	Low	Fixed	Hupixiong3
QTM-16	Missing parameter check in `stake::setUserLevel`	Logical	Low	Fixed	biakia
QTM-17	Potential divide by zero	Logical	Low	Fixed	biakia
QTM-18	Potential invalid IDO purchases	Logical	Low	Fixed	biakia
QTM-19	Unlimited while loop may cause gas exhaustion	DOS	Low	Acknowledged	biakia
QTM-20	Unupdated variable in `QEMOPair`	Code Style	Low	Fixed	Hupixiong3
QTM-21	Gas Optimization: Cache `userDynamicInfo[player]` to save gas	Gas Optimization	Informational	Fixed	biakia

QTM-22	Gas Optimization: Unnecessary `pairFor` function in `CnnLibrary::getReserves()`	Gas Optimization	Informational	Fixed	Hupixiong3
QTM-23	Gas Optimization:Cache array length outside for loop	Gas Optimization	Informational	Fixed	Hupixiong3, Xi_Zi
QTM-24	Gas Optimization:Redundant check	Gas Optimization	Informational	Declined	biakia
QTM-25	Gas Optimization:Unused Function	Gas Optimization	Informational	Fixed	Xi_Zi
QTM-26	Gas Optimization:Unused functions	Gas Optimization	Informational	Fixed	biakia, Hupixiong3
QTM-27	Gas Optimization:Unused variable	Gas Optimization	Informational	Fixed	infinityhacker, biakia, Hupixiong3, Xi_Zi
QTM-28	Gas Optimization:Use `calldata` instead of `memory` as much as possible	Gas Optimization	Informational	Fixed	biakia, Xi_Zi
QTM-29	Gas Optimization:Use immutable as much as possible	Language Specific	Informational	Acknowledged	biakia, Hupixiong3
QTM-30	Missing event record	Code Style	Informational	Fixed	Xi_Zi
QTM-31	Unlocked Pragma Version	Language Specific	Informational	Fixed	biakia

QTM-1:Incorrect unit in `QEMOStake::buyIDO()`

Category	Severity	Status	Contributor
Language Specific	Critical	Declined	Hupixiong3

Code Reference

- code/stake.sol#L428-L443

```
428:    function buyIDO(uint amount) external onlyEOA {
429:        require(block.timestamp < IDOEndTime, 'end');
430:        require(block.timestamp >= IDOStartTime, 'not start');
431:        require(amount <= IDOAmount, 'out of limit');
432:        require(userInfo[msg.sender].invitor != address(0), "not bond");
433:        usdt.transferFrom(msg.sender, address(this), amount);
434:        userIDO[msg.sender].amount += amount;
435:        IDOAmount -= amount;
436:        require(userIDO[msg.sender].amount >= 100 ether, "amount is too small");
437:        if (userIDO[msg.sender].isSuper) {
438:            require(userIDO[msg.sender].amount <= 3000 ether, "amount out of limit");
439:        } else {
440:            require(userIDO[msg.sender].amount <= 1000 ether, "amount out of limit");
441:        }
442:        emit BuyIDO(msg.sender, amount);
443:    }
```

Description

Hupixiong3 : The buyIDO function incorrectly assumes that the minimum amount for user participation in the IDO is in ether units. In the Solidity language, 1 ether is equivalent to 1e18, while the unit for USDT is 1e6. This discrepancy makes it almost impossible for users to participate in the IDO effectively.

Recommendation

Hupixiong3 : Use the correct unit conversion for proper calculation.

Client Response

Declined. We'll deploy the contract in BSC Chain, the decimal for USDT on BSC Chain is 1e18.

QTM-2:Integer Overflow

Category	Severity	Status	Contributor
DOS	Critical	Fixed	infinityhacker

Code Reference

- code/my_factory.sol#L515

```
515:                    uint outAmount = IERC20(token0).balanceOf(address(this)) - reserve0;
```

Description

infinityhacker : The contract is in version `0.5.16`, which does not have auto-integer-overflow protoection. But in swap function, there is a logic that directly sub two uint256 numbers without check, which will leads to integer overflow. Let me explain how it happens

```
            if (amount0Out > 0) {
                if (token0 != IQEMOFactory(factory).qemo()) {
                    _safeTransfer(_token0, to, amount0Out);
                    uint outAmount = IERC20(token1).balanceOf(address(this)) - reserve1;
                    if (outAmount > 0) {
                        _safeTransfer(token1, burnAddress, outAmount);
                    }
                } else {
                    _safeTransfer(_token0, to, amount0Out);

                }
                //                outAmount += amount0Out;
            }
            // optimistically transfer tokens
            if (amount1Out > 0) {
                if (token1 != IQEMOFactory(factory).qemo()) {
                    _safeTransfer(_token1, to, amount1Out);
                    uint outAmount = IERC20(token0).balanceOf(address(this)) - reserve0;
```

The code snippet above is the core of the swap function, according to the logic, when `amount0Out` or `amount1Out` > 0, `IERC20(token0/1).balanceOf(address(this)) - reserve1` will be sent to burn address, and corresponding token0/1 will be sent to user. So here comes into the bug, if `amount0Out` and `amount1Out` >0, because in the first `if`, the `_safeTransfer(_token0, to, amount0Out);` sends token0 to user, so currently balanceOf(token1) is less than reserve0, when enter the second `if` logic, the `IERC20(token0).balanceOf(address(this)) - reserve0;` will cause overflow, which will sent all tokens to

burn address. According to this, an malicious user can construct exact number using the pool balance, and send all tokens to burn address

PoC below

```
interface IQemoPool {
   function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
   function token0() external view returns (address);
   function token1() external view returns (address);
}

interface IERC20 {
    function balanceOf(address) external returns (uint256);
}

contract QemoHack {
    IQemoPool public pool;
    constructor(IQemoPool q) public {
            pool = q;
    }
    function hack() public {
        uint256 token0Balance = IERC20(pool.token0).balanceOf(address(pool));
        pool.swap(token0Balance, aMaliciousNumber, address(this), bytes(0));
    }
}
```

Recommendation

infinityhacker : Use Openzeppelin `.sub` instead

Client Response

Fixed.We used the Openzeppelin's `.sub` method.

QTM-3:Logical error:Use `referReward` instead of `reward`

Category	Severity	Status	Contributor
Logical	Critical	Fixed	biakia

Code Reference

- code/stake.sol#L413-L419

```
413:         uint referReward = userInfo[msg.sender].referReward;
414:         if (referReward > 0) {
415:             qemo.transferFrom(dynamicWallet, msg.sender, reward);
416:             userInfo[msg.sender].referReward = 0;
417:             claimInfo[msg.sender].push(ClaimInfo({types : 3, amount : referReward, timestamp : b
lock.timestamp}));
418:             emit ClaimReferReward(msg.sender, referReward);
419:         }
```

Description

biakia : In contract `QEMOStake`, the function `claimDynamicReward` is used to claim dynamic rewards. If the `referReward` is greater than 0, these rewards will also be claimed:

```
        uint referReward = userInfo[msg.sender].referReward;
        if (referReward > 0) {
            qemo.transferFrom(dynamicWallet, msg.sender, reward);
            userInfo[msg.sender].referReward = 0;
            claimInfo[msg.sender].push(ClaimInfo({types : 3, amount : referReward, timestamp : bloc
k.timestamp}));
            emit ClaimReferReward(msg.sender, referReward);
        }
```

However, the variable `reward` is used instead of `referReward` when calling `transferFrom` function, which results in the wrong number of referral rewards being claimed.

Recommendation

biakia : Consider below fix in the `claimDynamicReward()` function

```
        uint referReward = userInfo[msg.sender].referReward;
        if (referReward > 0) {
            qemo.transferFrom(dynamicWallet, msg.sender, referReward);
            userInfo[msg.sender].referReward = 0;
            claimInfo[msg.sender].push(ClaimInfo({types : 3, amount : referReward, timestamp : bloc
k.timestamp}));
            emit ClaimReferReward(msg.sender, referReward);
        }
```

Client Response

Fixed.We use `referReward` instead of `reward`.

QTM-4:Missing K check in swap function

Category	Severity	Status	Contributor
Logical	**Critical**	**Declined**	**infinityhacker, biakia, Xi_Zi**

Code Reference

- code/my_factory.sol#L485-L545
- code/my_factory.sol#L535
- code/my_factory.sol#L536-L542

```
485:     function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lo
ck {
486:         require(IQEMOFactory(factory).admin(msg.sender), 'invalid sender');
487:         require(amount0Out > 0 || amount1Out > 0, 'QEMO: INSUFFICIENT_OUTPUT_AMOUNT');
488:         (uint112 _reserve0, uint112 _reserve1,) = getReserves();
489:         // gas savings
490:         require(amount0Out < _reserve0 && amount1Out < _reserve1, 'QEMO: INSUFFICIENT_LIQUIDIT
Y');
491:
492:         uint balance0;
493:         uint balance1;
494:         {// scope for _token{0,1}, avoids stack too deep errors
495:             address _token0 = token0;
496:             address _token1 = token1;
497:             require(to != _token0 && to != _token1, 'QEMO: INVALID_TO');
498:             if (amount0Out > 0) {
499:                 if (token0 != IQEMOFactory(factory).qemo()) {
500:                     _safeTransfer(_token0, to, amount0Out);
501:                     uint outAmount = IERC20(token1).balanceOf(address(this)) - reserve1;
502:                     if (outAmount > 0) {
503:                         _safeTransfer(token1, burnAddress, outAmount);
504:                     }
505:                 } else {
506:                     _safeTransfer(_token0, to, amount0Out);
507:
508:                 }
509:                 //                outAmount += amount0Out;
510:             }
511:             // optimistically transfer tokens
512:             if (amount1Out > 0) {
513:                 if (token1 != IQEMOFactory(factory).qemo()) {
514:                     _safeTransfer(_token1, to, amount1Out);
515:                     uint outAmount = IERC20(token0).balanceOf(address(this)) - reserve0;
516:                     if (outAmount > 0) {
517:                         _safeTransfer(token0, burnAddress, outAmount);
518:                     }
519:                 } else {
520:                     _safeTransfer(_token1, to, amount1Out);
521:
```

```
522:
523:                 }
524:             }
525:             // optimistically transfer tokens
526:             if (data.length > 0) {
527:
528:             }
529:             balance0 = getToken0Blance();
530:             balance1 = IERC20(_token1).balanceOf(address(this));
531:         }
532:
533:
534:         uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out)
: 0;
535:         uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out)
: 0;
536:         //        require(amount0In > 0 || amount1In > 0, 'QEMO: INSUFFICIENT_INPUT_AMOUNT');
537:         //        {// scope for reserve{0,1}Adjusted, avoids stack too deep errors
538:         //            uint balance0Adjusted = (balance0.mul(10000).sub(amount0In.mul(25)));
539:         //            uint balance1Adjusted = (balance1.mul(10000).sub(amount1In.mul(25)));
540:         //            require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_res
erve1).mul(10000 ** 2), 'QEMO: K');
541:         //        }
542:
543:         _update(balance0, balance1, _reserve0, _reserve1);
544:         emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
545:     }
```

```
535:         uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out)
: 0;

536:         //        require(amount0In > 0 || amount1In > 0, 'QEMO: INSUFFICIENT_INPUT_AMOUNT');
537:         //        {// scope for reserve{0,1}Adjusted, avoids stack too deep errors
538:         //            uint balance0Adjusted = (balance0.mul(10000).sub(amount0In.mul(25)));
539:         //            uint balance1Adjusted = (balance1.mul(10000).sub(amount1In.mul(25)));
540:         //            require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_res
erve1).mul(10000 ** 2), 'QEMO: K');
```

```
541:         //         }
542:
```

Description

infinityhacker : Missing K verification in Pool contract in swap, when leads to the lost of whole pool asset According to the logic, after swap, the function didn't check `K` at the end of the swap, because qemo is forked from Uniswap V2, which also use **xy=k** to ensure correctly swap, without the check, an malicious user can drain all the tokens from pool Consider below POC contract

```
interface IQemoPool {
   function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
   function token0() external view returns (address);
   function token1() external view returns (address);
}

interface IERC20 {
    function balanceOf(address) external returns (uint256);
}

contract QemoHack {
    IQemoPool public pool;
    constructor(IQemoPool q) public {
            pool = q;
    }
    function hack() public {
        uint256 token0Balance = IERC20(pool.token0).balanceOf(address(pool));
        pool.swap(token0Balance, 0, address(this), bytes(0));
    }
}
```

biakia : In the function `swap` of `QEMOPair` contract, the check for `x*y>=k` is commented:

```
 uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) : 0;
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) :
0;
        //        require(amount0In > 0 || amount1In > 0, 'QEMO: INSUFFICIENT_INPUT_AMOUNT');
        //        {// scope for reserve{0,1}Adjusted, avoids stack too deep errors
        //            uint balance0Adjusted = (balance0.mul(10000).sub(amount0In.mul(25)));
        //            uint balance1Adjusted = (balance1.mul(10000).sub(amount1In.mul(25)));
        //            require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve
1).mul(10000 ** 2), 'QEMO: K');
        //        }
```

It will suffer from a well-known flashloan attack which has occurred in BurgerSwap. For more details, you can read this twitter thread from BugerSwap: **https://twitter.com/burger_cities/status/1398161871778115586**, or the analysis article: **https://quillhashteam.medium.com/burgerswap-flash-loan-attack-analysis-888b1911daef**

Xi_Zi : In the `swap` function of the `my_factory.sol` contract, specifically lines 536-542, there is a lack of K-value check. This omission can potentially lead to price manipulation, resulting in the depletion of pool funds or the emergence of inefficient trading or arbitrage opportunities. Although the function has the requirement of being called by an admin through the `require(IQEMOFactory(factory).admin(msg.sender), 'invalid sender');` statement, it cannot guarantee the integrity of the admin's actions. If the admin acts maliciously, it could potentially empty the pool.

```
function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
        require(IQEMOFactory(factory).admin(msg.sender), 'invalid sender');
        require(amount0Out > 0 || amount1Out > 0, 'QEMO: INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves();
        // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'QEMO: INSUFFICIENT_LIQUIDITY');

        uint balance0;
        uint balance1;
        {// scope for _token{0,1}, avoids stack too deep errors
            address _token0 = token0;
            address _token1 = token1;
            require(to != _token0 && to != _token1, 'QEMO: INVALID_TO');
            if (amount0Out > 0) {
                if (token0 != IQEMOFactory(factory).qemo()) {
                    _safeTransfer(_token0, to, amount0Out);
                    uint outAmount = IERC20(token1).balanceOf(address(this)) - reserve1;
                    if (outAmount > 0) {
                        _safeTransfer(token1, burnAddress, outAmount);
                    }
                } else {
                    _safeTransfer(_token0, to, amount0Out);

                }
                //                outAmount += amount0Out;
            }
            // optimistically transfer tokens
            if (amount1Out > 0) {
                if (token1 != IQEMOFactory(factory).qemo()) {
                    _safeTransfer(_token1, to, amount1Out);
                    uint outAmount = IERC20(token0).balanceOf(address(this)) - reserve0;
                    if (outAmount > 0) {
                        _safeTransfer(token0, burnAddress, outAmount);
                    }
                } else {
                    _safeTransfer(_token1, to, amount1Out);


                }
            }
            // optimistically transfer tokens
```

```
        if (data.length > 0) {

            }
            balance0 = getToken0Blance();
            balance1 = IERC20(_token1).balanceOf(address(this));
        }


        uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) :
0;
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) :
0;
        //        require(amount0In > 0 || amount1In > 0, 'QEMO: INSUFFICIENT_INPUT_AMOUNT');
        //        {// scope for reserve{0,1}Adjusted, avoids stack too deep errors
        //            uint balance0Adjusted = (balance0.mul(10000).sub(amount0In.mul(25)));
        //            uint balance1Adjusted = (balance1.mul(10000).sub(amount1In.mul(25)));
        //            require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve
1).mul(10000 ** 2), 'QEMO: K');
        //        }
        //
        _update(balance0, balance1, _reserve0, _reserve1);
        emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
    }
```

Recommendation

infinityhacker : Add K check

biakia : Consider adding the `x*y>=k` check in `swap` function.

Xi_Zi : To ensure the integrity of the Automated Market Maker (AMM) and maintain expected behavior, it is generally recommended to implement and enforce K-value checks in the `swap` function or other relevant parts of the contract. This check should be based on the token balances before and after the swap to verify if the K-value remains constant.

Client Response

Declined.Because all `QEMO` need to burn and `K` check will make `swap` call failed.

QTM-5:Vulnerability in `skim` Function of `my_factory.sol` Contract

Category	Severity	Status	Contributor
Flash Loan Attacks	Critical	Fixed	Xi_Zi

Code Reference

- code/my_factory.sol#L478-L482
- code/my_factory.sol#L549-L556

```
478:    function getBurnAmount() internal view returns (uint){
479:        uint balance0 = getToken0Blance();
480:        uint _burnAmount = balance0 - reserve0;
481:        return _burnAmount;
482:    }

549:    function skim(address to) external lock {
550:        address _token0 = token0;
551:        // gas savings
552:        address _token1 = token1;
553:        // gas savings
554:        _safeTransfer(_token0, to, (IERC20(_token0).balanceOf(address(this)).add(burnAmount)).su
b(reserve0));
555:        _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(this)).sub(reserve1));
556:    }
```

Description

Xi_Zi : In the `my_factory.sol` contract, there are issues in the `skim` function and the `getBurnAmount` function. The `skim` function has incorrect usage of the `burnAmount` variable. Instead of using an uninitialized variable, it should use the `getBurnAmount` function to calculate the burn amount. However, there is no assignment or initialization of `burnAmount` in the contract, and the `getBurnAmount` function is never called within the contract.

```
function skim(address to) external lock {
    address _token0 = token0;
    // gas savings
    address _token1 = token1;
    // gas savings
    _safeTransfer(_token0, to, (IERC20(_token0).balanceOf(address(this)).add(burnAmount)).sub(reserv
e0));
    _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(this)).sub(reserve1));
}

function getBurnAmount() internal view returns (uint){
    uint balance0 = getToken0Balance();
    uint _burnAmount = balance0 - reserve0;
    return _burnAmount;
}
```

Furthermore, the logic of the `skim` function is flawed. It transfers the token imbalances, including the burn amount, to the `to` address. This results in an unintended increase in token supply. An attacker could potentially exploit this vulnerability by using flash loans to borrow a large amount of USDT, purchasing QEMO tokens, transferring them to the pair contract, and then calling the `skim` function to retrieve the tokens along with the additional burn amount. The attacker can then sell all the QEMO tokens, repay the flash loan, and keep the extra QEMO tokens as profit.

Recommendation

Xi_Zi : 1. Modify the `skim` function to use the `getBurnAmount` function instead of the uninitialized `burnAmount` variable:

```
_safeTransfer(_token0, to, (IERC20(_token0).balanceOf(address(this)).add(getBurnAmount())).sub(reser
ve0));
```

2. Consider initializing or assigning a value to the `burnAmount` variable to ensure its proper usage throughout the contract.
3. Assess the logic of the `skim` function and ensure that token balances and reserves are handled correctly to prevent unintended increases or decreases in token supply.

Client Response

Fixed. According to recommendation, we deleted `burnAmount` .

QTM-6:Centralized risk.

Category	Severity	Status	Contributor
Logical	Medium	Fixed	Hupixiong3

Code Reference

- code/QEMO.sol#L96-L100

```
96:         if (!W[from] && !W[to]) {
97:             if (balanceOf(from) - amount < balanceLimit) {
98:                 amount = balanceOf(from) - balanceLimit;
99:             }
100:         }
```

Description

Hupixiong3 : In the _processTransfer function, there is a balance limit imposed when both the from and to addresses are not in the W mapping. When the balanceLimit is set too high, it can prevent users with lower asset balances from making transfers.

Recommendation

Hupixiong3 : When setting the balanceLimit, add a range interval to prevent it from being set too high. This will help prevent excessive balanceLimits that could hinder users with lower asset balances from making transfers.

Client Response

Fixed.We add more limit when setting `blanaceLimit`.

QTM-7:Logical error:IDO funds will be permanently locked in the contract

Category	Severity	Status	Contributor
Logical	**Medium**	**Fixed**	**biakia**

Code Reference

- code/stake.sol#L428-L478

```
428:     function buyIDO(uint amount) external onlyEOA {
429:         require(block.timestamp < IDOEndTime, 'end');
430:         require(block.timestamp >= IDOStartTime, 'not start');
431:         require(amount <= IDOAmount, 'out of limit');
432:         require(userInfo[msg.sender].invitor != address(0), "not bond");
433:         usdt.transferFrom(msg.sender, address(this), amount);
434:         userIDO[msg.sender].amount += amount;
435:         IDOAmount -= amount;
436:         require(userIDO[msg.sender].amount >= 100 ether, "amount is too small");
437:         if (userIDO[msg.sender].isSuper) {
438:             require(userIDO[msg.sender].amount <= 3000 ether, "amount out of limit");
439:         } else {
440:             require(userIDO[msg.sender].amount <= 1000 ether, "amount out of limit");
441:         }
442:         emit BuyIDO(msg.sender, amount);
443:     }
444:
445:
446:     function IdoStake() external checkStart checkRate onlyEOA {
447:         require(userIDO[msg.sender].amount > 0, "not buy");
448:         require(!userIDO[msg.sender].isStake, "already stake");
449:         uint amount = userIDO[msg.sender].amount;
450:         uint stakePower;
451:         if (userIDO[msg.sender].isSuper) {
452:             stakePower = userIDO[msg.sender].amount * 2;
453:         } else {
454:             stakePower = userIDO[msg.sender].amount * 15 / 10;
455:         }
456:
457:         uint _debt = countingDebt();
458:         if (userInfo[msg.sender].totalPower > 0) {
459:             userInfo[msg.sender].toClaim += _calculateReward(msg.sender);
460:         }
461:         userInfo[msg.sender].totalUAmount += amount;
462:         userInfo[msg.sender].totalPower += stakePower;
463:         userInfo[msg.sender].debt = _debt;
464:         totalUAmount += amount;
465:         totalPower += stakePower;
466:         debtInfo.debt = _debt;
467:         debtInfo.lastTime = block.timestamp;
468:         address invitor = userInfo[msg.sender].invitor;
469:         userInfo[invitor].referAmount += amount;
```

```
470:         if (userInfo[msg.sender].isFirst == false) {
471:             userInfo[msg.sender].isFirst = true;
472:             userInfo[invitor].refer_R ++;
473:         }
474:         _processReferAmount(msg.sender, amount);
475:         claimInfo[msg.sender].push(ClaimInfo({types : 2, amount : amount, timestamp : block.time
stamp}));
476:         userIDO[msg.sender].isStake = true;
477:         emit Stake(msg.sender, amount, stakePower);
478:     }
```

Description

biakia : In contract `QEMOStake`, user can participate in IDO by transferring `USDT` to the contract:

```
function buyIDO(uint amount) external onlyEOA {
        require(block.timestamp < IDOEndTime, 'end');
        require(block.timestamp >= IDOStartTime, 'not start');
        require(amount <= IDOAmount, 'out of limit');
        require(userInfo[msg.sender].invitor != address(0), "not bond");
        usdt.transferFrom(msg.sender, address(this), amount);
        ....
        ....
    }
```

The issue is that in the function `IdoStake`, these `USDT` will not be used to add liquidity. There is also no function to withdraw these `USDT`. All these `USDT` will be permanently locked in the contract.

Recommendation

biakia : Consider adding the IDO `USDT` into the liquidity in the `IdoStake` function:

```
function IdoStake() external checkStart checkRate onlyEOA {
       ...
       ...
       _processReferAmount(msg.sender, amount);
       _addLiquid(amount);
        claimInfo[msg.sender].push(ClaimInfo({types : 2, amount : amount, timestamp : block.timestam
p}));
        userIDO[msg.sender].isStake = true;
        emit Stake(msg.sender, amount, stakePower);
    }
```

Client Response

Fixed.We add `_addLiquid(amount);` to `IdoStake()` function.

QTM-8:Logical error:Missing check IDO Time Range in `stake:: setIDOTime`

Category	Severity	Status	Contributor
Logical	Medium	Fixed	Xi_Zi

Code Reference

- code/stake.sol#L199-L202

```
199:    function setIDOTime(uint startTime_, uint endTime_) external onlyOwner {
200:        IDOStartTime = startTime_;
201:        IDOEndTime = endTime_;
202:    }
```

Description

Xi_Zi : The `setIDOTime` function in the `stake.sol` contract, located at lines 199-202, should include a validation check to ensure that the provided `startTime` is less than the `endTime`. Without the proper validation check, it is possible for the `startTime` to be set greater than or equal to the `endTime`, which would result in an inconsistent IDO time range. This can lead to issues such as incorrect calculations, unexpected behavior during the IDO process, or even a potential denial-of-service (DoS) vulnerability.

Recommendation

Xi_Zi : To mitigate this risk, it is advised to include a validation check in the `setIDOTime` function to ensure that the `startTime` is less than the `endTime`. This can be done by adding a require statement as follows:

```
require(startTime_ < endTime_, "IDO: Invalid time range");
```

Client Response

Fixed. According to recommendation,we add the time check .

QTM-9:Logical error:Missing check startTime in `stake:: countingStakePower`

Category	Severity	Status	Contributor
Logical	Medium	Fixed	Xi_Zi

Code Reference

- code/stake.sol#L264-L280

```
264:    function countingStakePower(uint stakeAmount) public view returns (uint){
265:        if (startTime == 0) {
266:            return 0;
267:        }
268:        uint day = (block.timestamp - startTime) / 1 days;
269:        uint power = stakeAmount;
270:        while (true) {
271:            if (day > 5) {
272:                power = power * (1012 ** 5) / 1000 ** 5;
273:                day -= 5;
274:            } else {
275:                power = power * (1012 ** day) / 1000 ** day;
276:                break;
277:            }
278:        }
279:        return power;
280:    }
```

Description

Xi_Zi : In the `countingStakePower` function of the `stake.sol` contract, there is a high-risk issue in the condition check. The current condition `if (startTime == 0)` is incorrect and may lead to errors when calculating the `day` variable below. Instead, the condition should be `if (block.timestamp < startTime)` to properly handle the scenario when the current block timestamp is less than the start time.

Recommendation

Xi_Zi : Update the condition in the `countingStakePower` function to `if (block.timestamp < startTime)` to properly handle cases where the current block timestamp is less than the start time. This will prevent potential errors when calculating the `day` variable.

```
function countingStakePower(uint stakeAmount) public view returns (uint){
    if (block.timestamp < startTime) {
        return 0;
    }
    uint day = (block.timestamp - startTime) / 1 days;
    uint power = stakeAmount;
    while (true) {
        if (day > 5) {
            power = power * (1012 ** 5) / 1000 ** 5;
            day -= 5;
        } else {
            power = power * (1012 ** day) / 1000 ** day;
            break;
        }
    }
    return power;
}
```

Client Response

Fixed. According to recommendation,we fixed the time check .

QTM-10:Logical error:Referral rewards will be distributed twice in function `IdoStake`

Category	Severity	Status	Contributor
Logical	Medium	Fixed	biakia

Code Reference

- code/stake.sol#L446-L478

```
446:    function IdoStake() external checkStart checkRate onlyEOA {
447:        require(userIDO[msg.sender].amount > 0, "not buy");
448:        require(!userIDO[msg.sender].isStake, "already stake");
449:        uint amount = userIDO[msg.sender].amount;
450:        uint stakePower;
451:        if (userIDO[msg.sender].isSuper) {
452:            stakePower = userIDO[msg.sender].amount * 2;
453:        } else {
454:            stakePower = userIDO[msg.sender].amount * 15 / 10;
455:        }
456:
457:        uint _debt = countingDebt();
458:        if (userInfo[msg.sender].totalPower > 0) {
459:            userInfo[msg.sender].toClaim += _calculateReward(msg.sender);
460:        }
461:        userInfo[msg.sender].totalUAmount += amount;
462:        userInfo[msg.sender].totalPower += stakePower;
463:        userInfo[msg.sender].debt = _debt;
464:        totalUAmount += amount;
465:        totalPower += stakePower;
466:        debtInfo.debt = _debt;
467:        debtInfo.lastTime = block.timestamp;
468:        address invitor = userInfo[msg.sender].invitor;
469:        userInfo[invitor].referAmount += amount;
470:        if (userInfo[msg.sender].isFirst == false) {
471:            userInfo[msg.sender].isFirst = true;
472:            userInfo[invitor].refer_R ++;
473:        }
474:        _processReferAmount(msg.sender, amount);
475:        claimInfo[msg.sender].push(ClaimInfo({types : 2, amount : amount, timestamp : block.time
stamp}));
476:        userIDO[msg.sender].isStake = true;
477:        emit Stake(msg.sender, amount, stakePower);
478:    }
```

Description

biakia : In contract `QEMOStake`, the function `IdoStake()` is used for IDO users to stake their powers. It will also assign referral rewards to the invitor of the `msg.sender`:

```
        address invitor = userInfo[msg.sender].invitor;
        userInfo[invitor].referAmount += amount;
        if (userInfo[msg.sender].isFirst == false) {
            userInfo[msg.sender].isFirst = true;
            userInfo[invitor].refer_R ++;
        }
        _processReferAmount(msg.sender, amount);
```

The issue here is that the same amount of referral reward is also assigned to the invitor in the function `_processReferAmount`:

```
function _processReferAmount(address player, uint amount) internal {
        address invitor = userInfo[player].invitor;
        while (true) {
            if (invitor == address(0) || invitor == address(this)) {
                break;
            }
            UserInfo storage info = userInfo[invitor];
            UserDynamicInfo storage dynamicInfo = userDynamicInfo[invitor];
            info.referAmount += amount;
            {
                uint level = dynamicInfo.level;
                uint newLevel = countingUserLevel(invitor);
                if (newLevel > level) {
                    dynamicInfo.level = newLevel;
                    if (level != 0) {
                        userDynamicInfo[info.invitor].levelReferAmount[level] --;
                    }
                    userDynamicInfo[info.invitor].levelReferAmount[newLevel] ++;
                }

            }

            invitor = userInfo[invitor].invitor;
        }
    }
```

Here we can see that the input param `player` is `msg.sender` and the code `address invitor = userInfo[player].invitor;` will be `address invitor = userInfo[msg.sender].invitor;`. This means the invitor in `IdoStake` is the same with the invitor in `_processReferAmount`. In the `while` loop, this invitor is once again assigned referral rewards.

Recommendation

biakia : Consider removing the referral rewards in the `IdoStake()` function

```
        address invitor = userInfo[msg.sender].invitor;
        if (userInfo[msg.sender].isFirst == false) {
            userInfo[msg.sender].isFirst = true;
            userInfo[invitor].refer_R ++;
        }
        _processReferAmount(msg.sender, amount);
```

Client Response

Fixed. According to recommendation, we remove `refeerral rewards` from `IdoStake()` function.

QTM-11:Logical error:Some usdt may be permanently locked after swapping in the contract `QEMOStake`

Category	Severity	Status	Contributor
Logical	Medium	Fixed	biakia

Code Reference

- code/stake.sol#L303-L341

```
303:     function _stake(address player, uint amount) internal {
304:         require(amount > 0, "amount is zero");
305:         require(usdt.balanceOf(player) >= amount, "usdt balance not enough");
306:         usdt.transferFrom(msg.sender, address(this), amount);
307:         uint stakePower = countingStakePower(amount);
308:         uint _debt = countingDebt();
309:         if (userInfo[player].totalPower > 0) {
310:             userInfo[player].toClaim += _calculateReward(player);
311:         }
312:
313:         userInfo[player].totalUAmount += amount;
314:         userInfo[player].totalPower += stakePower;
315:         userInfo[player].debt = _debt;
316:         totalUAmount += amount;
317:         totalPower += stakePower;
318:         debtInfo.debt = _debt;
319:         debtInfo.lastTime = block.timestamp;
320:         address invitor = userInfo[player].invitor;
321:         if (userInfo[player].isFirst == false) {
322:             userInfo[player].isFirst = true;
323:             userInfo[invitor].refer_R ++;
324:         }
325:         _processReferAmount(player, amount);
326:         _addLiquid(amount);
327:         claimInfo[player].push(ClaimInfo({types : 2, amount : amount, timestamp : block.timestam
p}));
328:         emit Stake(player, amount, stakePower);
329:     }
330:
331:     function _addLiquid(uint amount) internal {
332:         usdt.approve(address(router), amount);
333:         uint lastToken = qemo.balanceOf(address(this));
334:         address[] memory path = new address[](2);
335:         path[0] = address(usdt);
336:         path[1] = address(qemo);
337:         router.swapExactTokensForTokensSupportingFeeOnTransferTokens(amount / 2, 0, path, addres
s(this), block.timestamp + 721);
338:         uint tokenAmount = qemo.balanceOf(address(this)) - lastToken;
339:         qemo.approve(address(router), tokenAmount);
340:         router.addLiquidity(address(usdt), address(qemo), amount / 2, tokenAmount, 0, 0, address
(this), block.timestamp + 721);
341:     }
```

Description

biakia : In contract `QEMOStake`, user can buy `stakePower` through `USDT`:

```
function _stake(address player, uint amount) internal {
        require(amount > 0, "amount is zero");
        require(usdt.balanceOf(player) >= amount, "usdt balance not enough");
        usdt.transferFrom(msg.sender, address(this), amount);
        uint stakePower = countingStakePower(amount);
        ....
        ....
        _addLiquid(amount);
```

These `USDT` will be used to add liquidity through `_addLiquid` function:

```
function _addLiquid(uint amount) internal {
        usdt.approve(address(router), amount);
        uint lastToken = qemo.balanceOf(address(this));
        address[] memory path = new address[](2);
        path[0] = address(usdt);
        path[1] = address(qemo);
        router.swapExactTokensForTokensSupportingFeeOnTransferTokens(amount / 2, 0, path, address(th
is), block.timestamp + 721);
        uint tokenAmount = qemo.balanceOf(address(this)) - lastToken;
        qemo.approve(address(router), tokenAmount);
        router.addLiquidity(address(usdt), address(qemo), amount / 2, tokenAmount, 0, 0, address(thi
s), block.timestamp + 721);
    }
```

It will use half of the `USDT` to buy `QEMO` and then use the another half to add liquidity to the `USDT-QEMO` pair. In contract `QEMO`, it will take some fees when transferring to the `USDT-QEMO` pair:

```
function _processTransfer(address from, address to, uint amount) internal {
        if (!W[from] && !W[to]) {
            if (balanceOf(from) - amount < balanceLimit) {
                amount = balanceOf(from) - balanceLimit;
            }
        }

        if (isFree()) {
            _transfer(from, to, amount);
            return;
        }
        if (pairs[from] || pairs[to]) {
            if (W[from] || W[to]) {
                _transfer(from, to, amount);
            } else {
                _processPancakeFee(from, to, amount);
            }
            return;
        }
        if (to == swap || from == swap) {
            _processInsideFee(from, to, amount);
            return;
        }
        _transfer(from, to, amount);

    }

function _processPancakeFee(address from, address to, uint amount) internal {
        uint fee = amount * pancakeFee / 100;
        _transfer(from, burnAddress, fee / 2);
        _transfer(from, fund, fee * 3 / 10);
        _transfer(from, group, fee * 2 / 10);
        _transfer(from, to, amount - fee);
    }
```

That means the actual amount of `QEMO` the pair received is smaller than the param `amount`. Now let's say Bob pays 1000 `USDT` to buy `stakePower`, and 500 `USDT` will be swapped to `QEMO` and let's say the price of `QEMO` now is 1 `USDT` and the `pancakeFee` is 2%. After calling `swapExactTokensForTokensSupportingFeeOnTransferTokens`, the contract `QEMOStake` will receive 490 `QEMO`. And then, it will use 500 `USDT` and 490 `QEMO` to add liquidity. In the function `addLiquidity`, it will recalculate the amount of `USDT` based on the amount of `QEMO`:

```
function _addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin
    ) internal virtual returns (uint amountA, uint amountB) {
        // create the pair if it doesn't exist yet
        if (ICnnFactory(factory).getPair(tokenA, tokenB) == address(0)) {
            ICnnFactory(factory).createPair(tokenA, tokenB);
        }
        (uint reserveA, uint reserveB) = CnnLibrary.getReserves(factory, tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint amountBOptimal = CnnLibrary.quote(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(amountBOptimal >= amountBMin, 'CnnRouter: INSUFFICIENT_B_AMOUNT');
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint amountAOptimal = CnnLibrary.quote(amountBDesired, reserveB, reserveA);
                assert(amountAOptimal <= amountADesired);
                require(amountAOptimal >= amountAMin, 'CnnRouter: INSUFFICIENT_A_AMOUNT');
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }
```

That means the actual amount of `USDT` to be used to add liquidity will be smaller than 500 `USDT`. At last, some of `USDT` will be remaining in the contract `QEMOStake`. What's more, there is no function to withdraw these `USDT`. All the `USDT` remaining in the `QEMOStake` will be permanently locked.

Recommendation

biakia : Consider providing a function to withdraw these remaining `USDT`.

Client Response

Fixed. According to recommendation, we add the `USDT` withdraw method.

QTM-12:Invalid Balance Checks

Category	Severity	Status	Contributor
Logical	Low	Fixed	biakia

Code Reference

- code/stake.sol#L303-L329

```
303:    function _stake(address player, uint amount) internal {
304:        require(amount > 0, "amount is zero");
305:        require(usdt.balanceOf(player) >= amount, "usdt balance not enough");
306:        usdt.transferFrom(msg.sender, address(this), amount);
307:        uint stakePower = countingStakePower(amount);
308:        uint _debt = countingDebt();
309:        if (userInfo[player].totalPower > 0) {
310:            userInfo[player].toClaim += _calculateReward(player);
311:        }
312:
313:        userInfo[player].totalUAmount += amount;
314:        userInfo[player].totalPower += stakePower;
315:        userInfo[player].debt = _debt;
316:        totalUAmount += amount;
317:        totalPower += stakePower;
318:        debtInfo.debt = _debt;
319:        debtInfo.lastTime = block.timestamp;
320:        address invitor = userInfo[player].invitor;
321:        if (userInfo[player].isFirst == false) {
322:            userInfo[player].isFirst = true;
323:            userInfo[invitor].refer_R ++;
324:        }
325:        _processReferAmount(player, amount);
326:        _addLiquid(amount);
327:        claimInfo[player].push(ClaimInfo({types : 2, amount : amount, timestamp : block.timestam
p}));
328:        emit Stake(player, amount, stakePower);
329:    }
```

Description

biakia : In contract `QEMOStake`, the function `_stake` will check whether there are enough `USDT` in the player's wallet:

```
function _stake(address player, uint amount) internal {
        require(amount > 0, "amount is zero");
        require(usdt.balanceOf(player) >= amount, "usdt balance not enough");
        usdt.transferFrom(msg.sender, address(this), amount);
        ...
        ...
```

The function `stakeFor` will call function `_stake` and the `player` can be different from `msg.sender`:

```
function stakeFor(address player, uint amount) external checkStart checkRate {
        require(msg.sender == address(router), "not router");
        //        require(amount >= 100 ether, "amount is too small");
        if (userInfo[player].invitor == address(0)) {
            userInfo[player].invitor = address(this);
            emit Bond(player, address(this));
        }
        _stake(player, amount);
    }
```

Here we say that the `player` is different from `msg.sender`, the `player` has no `USDT` but the `msg.sender` has enough `USDT`. The `stakeFor` should succeed because the `msg.sender` has enough `USDT`, but it actually will fail because the check `require(usdt.balanceOf(player) >= amount, "usdt balance not enough");` will revert.

Recommendation

biakia : Consider below fix in the `_stake()` function

```
function _stake(address player, uint amount) internal {
        require(amount > 0, "amount is zero");
        require(usdt.balanceOf(msg.sender) >= amount, "usdt balance not enough");
        usdt.transferFrom(msg.sender, address(this), amount);
        ...
        ...
```

Client Response

Fixed. According to recommendation,we remove balance check.

QTM-13:Logical error:Missing update `rate` after `dailyOut` changed

Category	Severity	Status	Contributor
Logical	Low	Declined	biakia

Code Reference

- code/stake.sol#L133-L166

```
133:    modifier checkRate(){
134:        if (!isSetArr[0] && !isSetArr[1] && !isTopDaily) {
135:            debtInfo.debt = countingDebt();
136:            debtInfo.lastTime = block.timestamp;
137:            uint day = (block.timestamp - startTime) / 1 days;
138:            uint out;
139:            uint _dailyOut = totalOutAmount * 2 / 1000;
140:            while (true) {
141:                if (day > 5) {
142:                    dailyOut = _dailyOut * (110 ** 5) / 100 ** 5;
143:                    rate = dailyOut / 86400;
144:                    day -= 5;
145:                } else {
146:                    dailyOut = _dailyOut * (110 ** day) / 100 ** day;
147:                    rate = dailyOut / 86400;
148:                    break;
149:                }
150:            }
151:            if (dailyOut >= totalOutAmount / 100) {
152:                dailyOut = totalOutAmount / 100;
153:                isTopDaily = true;
154:            }
155:        }
156:        uint stageDay = (block.timestamp - stageNextClaimTime) / 1 days;
157:        if (stageDay > 1) {
158:            if (stageRate == 0) {
159:                stageRate = 5;
160:            }
161:            uint reward = stageDay * dailyOut * stageRate / 100;
162:            qemo.transfer(stage, reward);
163:            stageNextClaimTime += stageDay * 1 days;
164:        }
165:        _;
166:    }
```

Description

biakia : In contract `QEMOStake`, the modifier `checkRate` will reset `dailyOut` and `rate` when the `isTopDaily` is false:

```
        if (!isSetArr[0] && !isSetArr[1] && !isTopDaily) {
            debtInfo.debt = countingDebt();
            debtInfo.lastTime = block.timestamp;
            uint day = (block.timestamp - startTime) / 1 days;
            uint out;
            uint _dailyOut = totalOutAmount * 2 / 1000;
            while (true) {
                if (day > 5) {
                    dailyOut = _dailyOut * (110 ** 5) / 100 ** 5;
                    rate = dailyOut / 86400;
                    day -= 5;
                } else {
                    dailyOut = _dailyOut * (110 ** day) / 100 ** day;
                    rate = dailyOut / 86400;
                    break;
                }
            }
            if (dailyOut >= totalOutAmount / 100) {
                dailyOut = totalOutAmount / 100;
                isTopDaily = true;
            }
        }
```

If `dailyOut` is greater than `totalOutAmount / 100`, the `dailyOut` will be reset to `totalOutAmount / 100`:

```
            if (dailyOut >= totalOutAmount / 100) {
                dailyOut = totalOutAmount / 100;
                isTopDaily = true;
            }
```

The issue here is that the `rate` is not updated after the `dailyOut` changed. This will result in a larger actual `rate` than expected and the number of daily rewards calculated based on `rate` will be larger than `dailyOut`.

Recommendation

biakia : Consider below fix in the `checkRate()` function

```
if (!isSetArr[0] && !isSetArr[1] && !isTopDaily) {
            debtInfo.debt = countingDebt();
            debtInfo.lastTime = block.timestamp;
            uint day = (block.timestamp - startTime) / 1 days;
            uint out;
            uint _dailyOut = totalOutAmount * 2 / 1000;
            while (true) {
                if (day > 5) {
                    dailyOut = _dailyOut * (110 ** 5) / 100 ** 5;
                    rate = dailyOut / 86400;
                    day -= 5;
                } else {
                    dailyOut = _dailyOut * (110 ** day) / 100 ** day;
                    rate = dailyOut / 86400;
                    break;
                }
            }
            if (dailyOut >= totalOutAmount / 100) {
                dailyOut = totalOutAmount / 100;
                rate = dailyOut / 86400;
                isTopDaily = true;
            }
        }
```

Client Response

Declined.We remove modifier `checkRate()`.

QTM-14:Logical error:Missing update `refer_R` of the invitor when calling function `stakeFor`

Category	Severity	Status	Contributor
Logical	Low	Declined	biakia

Code Reference

- code/stake.sol#L349-L357

```
349:    function stakeFor(address player, uint amount) external checkStart checkRate {
350:        require(msg.sender == address(router), "not router");
351:        //        require(amount >= 100 ether, "amount is too small");
352:        if (userInfo[player].invitor == address(0)) {
353:            userInfo[player].invitor = address(this);
354:            emit Bond(player, address(this));
355:        }
356:        _stake(player, amount);
357:    }
```

Description

biakia : In contract `QEMOStake`, the function `stakeFor` will bind a default invitor for the user if the user does not have any invitor:

```
function stakeFor(address player, uint amount) external checkStart checkRate {
        require(msg.sender == address(router), "not router");
        //        require(amount >= 100 ether, "amount is too small");
        if (userInfo[player].invitor == address(0)) {
            userInfo[player].invitor = address(this);
            emit Bond(player, address(this));
        }
        _stake(player, amount);
    }
```

The issue here is that the `refer_R` of the invitor is not updated, which is different from the function `bond`:

```
function bond(address invitor) external onlyEOA {
        ...
        ...
        userInfo[msg.sender].invitor = invitor;
        userInfo[invitor].refer_n ++;
        emit Bond(msg.sender, invitor);
    }
```

Recommendation

biakia : Consider below fix in the `stakeFor()` function

```
function stakeFor(address player, uint amount) external checkStart checkRate {
        require(msg.sender == address(router), "not router");
        //        require(amount >= 100 ether, "amount is too small");
        if (userInfo[player].invitor == address(0)) {
            userInfo[player].invitor = address(this);
            userInfo[address(this)].refer_n ++;
            emit Bond(player, address(this));
        }
        _stake(player, amount);
    }
```

Client Response

Declined.Contract address don’t need refer_n

QTM-15:Missing 0 address check in `QEMOFactory`::`setFeeToSetter()` and `CnnRouter`::`changeOwner()`

Category	Severity	Status	Contributor
Logical	Low	Fixed	Hupixiong3

Code Reference

- code/router.sol#L466-L469
- code/my_factory.sol#L631-L634

```
466:    function changeOwner(address newOwner) public {
467:        require(msg.sender == owner, "Only owner can change ownership");
468:        owner = newOwner;
469:    }

631:    function setFeeToSetter(address _feeToSetter) external {
632:        require(msg.sender == feeToSetter, 'QEMO: FORBIDDEN');
633:        feeToSetter = _feeToSetter;
634:    }
```

Description

Hupixiong3 : The function setFeeToSetter is missing a 0 address check. If feeToSetter is set to the 0 address, it will be impossible to modify, and functions that require the feeToSetter address to be called will not work properly.
Hupixiong3 : The changeOwner function is missing a 0 address check.

Recommendation

Hupixiong3 : Add a 0 address check.
Hupixiong3 : Add a 0 address check.

Client Response

Fixed.We add 0 address check.

QTM-16:Missing parameter check in `stake::setUserLevel`

Category	Severity	Status	Contributor
Logical	Low	Fixed	biakia

Code Reference

- code/stake.sol#L480-L489

```
480:    function setUserLevel(address addr, uint level) external onlyOwner {
481:        address invitor = userInfo[addr].invitor;
482:        uint oldLevel = userDynamicInfo[addr].level;
483:        userDynamicInfo[addr].level = level;
484:        levelSet[addr] = level;
485:        if (oldLevel != 0) {
486:            userDynamicInfo[invitor].levelReferAmount[oldLevel] --;
487:        }
488:        userDynamicInfo[invitor].levelReferAmount[level] ++;
489:    }
```

Description

biakia : The function `setUserLevel` is a privileged function and can reset the value of `userDynamicInfo[addr].level`:

```
function setUserLevel(address addr, uint level) external onlyOwner {
        address invitor = userInfo[addr].invitor;
        uint oldLevel = userDynamicInfo[addr].level;
        userDynamicInfo[addr].level = level;
        levelSet[addr] = level;
        if (oldLevel != 0) {
            userDynamicInfo[invitor].levelReferAmount[oldLevel] --;
        }
        userDynamicInfo[invitor].levelReferAmount[level] ++;
}
```

In the function `_processDynamic`, the value of `userDynamicInfo[addr].level` will be used to read rate from the array `dynamicRate`:

```
                UserDynamicInfo storage dynamicInfo = userDynamicInfo[invitor];
                currentLevel = dynamicInfo.level;
                if (currentLevel < lastLevel) {
                    invitor = userInfo[invitor].invitor;
                    continue;
                }
                if (currentLevel > lastLevel) {
                    uint rewardRate = dynamicRate[currentLevel] - dynamicRate[lastLevel];
                    ....
                    ....
```

In function `initialize`, the `dynamicRate` is an array that has 8 elements:

```
function initialize() initializer public {
        ....
        dynamicRate = [0, 1875, 3125, 4250, 5250, 6125, 6875, 7500];
        ....
    }
```

The issue here is that there is no check for the `level` in function `setUserLevel`. If the `level` is set as 9, then the `userDynamicInfo[addr].level` will be 9 and the `currentLevel` in function `_processDynamic` will be 9. The `dynamicRate[currentLevel]` will be `dynamicRate[9]` and it will encounter an array-out-of-bounds error. At last, this user will be unable to call `claimReward` due to the array-out-of-bounds error occurred in function `_processDynamic`.

Recommendation

biakia : Consider below fix in the `setUserLevel` function

```
function setUserLevel(address addr, uint level) external onlyOwner {
        require(level<8,"invalid level");
        address invitor = userInfo[addr].invitor;
        uint oldLevel = userDynamicInfo[addr].level;
        userDynamicInfo[addr].level = level;
        levelSet[addr] = level;
        if (oldLevel != 0) {
            userDynamicInfo[invitor].levelReferAmount[oldLevel] --;
        }
        userDynamicInfo[invitor].levelReferAmount[level] ++;
    }
```

Client Response

Fixed.We add level check.

QTM-17:Potential divide by zero

Category	Severity	Status	Contributor
Logical	Low	Fixed	biakia

Code Reference

- code/QEMO.sol#L74-L78

```
74:     function getTokenPrice() public view returns(uint){
75:         uint uBalance = usdt.balanceOf(swap);
76:         uint tBalance = balanceOf(swap);
77:         return uBalance * 1e18 / tBalance;
78:     }
```

Description

biakia : In contract `Quantemo`, the function `getTokenPrice` will calculate the price based on the balance of `swap`:

```
function getTokenPrice() public view returns(uint){
        uint uBalance = usdt.balanceOf(swap);
        uint tBalance = balanceOf(swap);
        return uBalance * 1e18 / tBalance;
    }
```

It is possible that the `tBalance` is zero. The call will revert without corresponding error information when the `tBalance` is 0.

Recommendation

biakia : Recommend adding check to give the corresponding error information.

Consider below fix in the `getTokenPrice()` function

```
function getTokenPrice() public view returns(uint){
        uint uBalance = usdt.balanceOf(swap);
        uint tBalance = balanceOf(swap);
        require(tBalance>0,"tBalance is 0");
        return uBalance * 1e18 / tBalance;
}
```

Client Response

Fixed.We add tBalance check.

QTM-18:Potential invalid IDO purchases

Category	Severity	Status	Contributor
Logical	Low	Fixed	biakia

Code Reference

- code/stake.sol#L428-L478

```
428:     function buyIDO(uint amount) external onlyEOA {
429:         require(block.timestamp < IDOEndTime, 'end');
430:         require(block.timestamp >= IDOStartTime, 'not start');
431:         require(amount <= IDOAmount, 'out of limit');
432:         require(userInfo[msg.sender].invitor != address(0), "not bond");
433:         usdt.transferFrom(msg.sender, address(this), amount);
434:         userIDO[msg.sender].amount += amount;
435:         IDOAmount -= amount;
436:         require(userIDO[msg.sender].amount >= 100 ether, "amount is too small");
437:         if (userIDO[msg.sender].isSuper) {
438:             require(userIDO[msg.sender].amount <= 3000 ether, "amount out of limit");
439:         } else {
440:             require(userIDO[msg.sender].amount <= 1000 ether, "amount out of limit");
441:         }
442:         emit BuyIDO(msg.sender, amount);
443:     }
444:
445:
446:     function IdoStake() external checkStart checkRate onlyEOA {
447:         require(userIDO[msg.sender].amount > 0, "not buy");
448:         require(!userIDO[msg.sender].isStake, "already stake");
449:         uint amount = userIDO[msg.sender].amount;
450:         uint stakePower;
451:         if (userIDO[msg.sender].isSuper) {
452:             stakePower = userIDO[msg.sender].amount * 2;
453:         } else {
454:             stakePower = userIDO[msg.sender].amount * 15 / 10;
455:         }
456:
457:         uint _debt = countingDebt();
458:         if (userInfo[msg.sender].totalPower > 0) {
459:             userInfo[msg.sender].toClaim += _calculateReward(msg.sender);
460:         }
461:         userInfo[msg.sender].totalUAmount += amount;
462:         userInfo[msg.sender].totalPower += stakePower;
463:         userInfo[msg.sender].debt = _debt;
464:         totalUAmount += amount;
465:         totalPower += stakePower;
466:         debtInfo.debt = _debt;
467:         debtInfo.lastTime = block.timestamp;
468:         address invitor = userInfo[msg.sender].invitor;
469:         userInfo[invitor].referAmount += amount;
```

```
470:         if (userInfo[msg.sender].isFirst == false) {
471:             userInfo[msg.sender].isFirst = true;
472:             userInfo[invitor].refer_R ++;
473:         }
474:         _processReferAmount(msg.sender, amount);
475:         claimInfo[msg.sender].push(ClaimInfo({types : 2, amount : amount, timestamp : block.time
stamp}));
476:         userIDO[msg.sender].isStake = true;
477:         emit Stake(msg.sender, amount, stakePower);
478:     }
```

Description

biakia : In contract `QEMOStake`, users can call the function `buyIDO` to buy power and then call the function `IdoStake` to stake their power. The function `IdoStake` can only be called once due to the following check:

```
require(!userIDO[msg.sender].isStake, "already stake");
```

It is possible that users still buy IDO after they stake their power. However, these power can not be staked again because the function `IdoStake` has already been called. At last, these IDO purchases will be invalid.

Recommendation

biakia : Consider checking whether the user has already staked in function `buyIDO`:

```
function buyIDO(uint amount) external onlyEOA {
        require(block.timestamp < IDOEndTime, 'end');
        require(block.timestamp >= IDOStartTime, 'not start');
        require(!userIDO[msg.sender].isStake, "already stake");
        ...
        ...
```

Client Response

Fixed.We add the check to `buyIDO` function.

QTM-19:Unlimited while loop may cause gas exhaustion

Category	Severity	Status	Contributor
DOS	Low	Acknowledged	biakia

Code Reference

- code/stake.sol#L550-L605

```
550:     function _processDynamic(address player, uint totalReward) internal {
551:         address invitor = userInfo[player].invitor;
552:         if (invitor == address(0) || invitor == address(this)) {
553:             return;
554:         }
555:
556:         //        uint left = totalReward;
557:         {
558:             address[] memory lists = new address[](100);
559:             uint lastLevel = 0;
560:             uint currentLevel;
561:             uint index;
562:             uint tempReward;
563:             uint sameLevel;
564:             while (true) {
565:                 if (invitor == address(0) || invitor == address(this)) {
566:                     if (index > 0) {
567:                         tempReward = totalReward * sameRate[sameLevel - 4] / 10000 / (index);
568:                         for (uint i = 0; i < index + 1; i++) {
569:                             userDynamicInfo[lists[i]].sameToClaim += tempReward;
570:                         }
571:                     }
572:                     break;
573:                 }
574:                 UserDynamicInfo storage dynamicInfo = userDynamicInfo[invitor];
575:                 currentLevel = dynamicInfo.level;
576:                 if (currentLevel < lastLevel) {
577:                     invitor = userInfo[invitor].invitor;
578:                     continue;
579:                 }
580:                 if (currentLevel > lastLevel) {
581:                     uint rewardRate = dynamicRate[currentLevel] - dynamicRate[lastLevel];
582:                     tempReward = totalReward * rewardRate / 10000;
583:                     userDynamicInfo[invitor].normalToClaim += tempReward;
584:                     lastLevel = dynamicInfo.level;
585:                     if (index > 0) {
586:                         tempReward = totalReward * sameRate[sameLevel - 4] / 10000 / (index);
587:                         for (uint i = 0; i < index + 1; i++) {
588:                             userDynamicInfo[lists[i]].sameToClaim += tempReward;
```

```
589:                         }
590:                         index = 0;
591:                         lists = new address[](100);
592:                     }
593:
594:                 } else if (currentLevel == lastLevel && currentLevel >= 4) {
595:                     sameLevel = currentLevel;
596:                     lists[index] = invitor;
597:                     index ++;
598:                     //                     tempReward = totalReward * sameRate[currentLevel - 4]
/ 10000 / (index + 1);
599:                 }
600:                 invitor = userInfo[invitor].invitor;
601:             }
602:         }
603:
604:
605:     }
```

Description

biakia : In contract `QEMOStake`, the function `_processDynamic` will distribute referral rewards to all invitors through a `while` loop. The condition `while (true)` means the loop is unlimited. It may cause gas exhaustion and the contract cannot be used normally.

Recommendation

biakia : Consider setting an upper limit for the number of the loop.

Client Response

Acknowledged.

QTM-20:Unupdated variable in `QEMOPair`

Category	Severity	Status	Contributor
Code Style	Low	Fixed	Hupixiong3

Code Reference

- code/my_factory.sol#L554

```
554:        _safeTransfer(_token0, to, (IERC20(_token0).balanceOf(address(this)).add(burnAmount)).su
b(reserve0));
```

Description

Hupixiong3 : In the "skim" function, there is a safetransfer involving the "burnAmount" variable, but it is never updated and remains in its initial state of 0.

Recommendation

Hupixiong3 : Delete the variable or add the necessary functionality to update it.

Client Response

Fixed.We remove the `burnAmount`.

QTM-21:Gas Optimization: Cache `userDynamicInfo[player]` to save gas

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	biakia

Code Reference

- code/stake.sol#L491-L522

```
491:     function countingUserLevel(address player) public view returns (uint){
492:         UserInfo storage info = userInfo[player];
493:         uint referAmount = info.referAmount;
494:         uint refer_R = info.refer_R;
495:         uint level = userDynamicInfo[player].level;
496:         uint outLevel;
497:         if (levelSet[player] > 0) {
498:             return levelSet[player];
499:         }
500:         if (level < 1) {
501:             if (referAmount >= 30000 ether && refer_R >= 5) {
502:                 outLevel = 1;
503:             }
504:         }
505:         if (outLevel >= 1 || level >= 1) {
506:             if (userDynamicInfo[player].levelReferAmount[6] >= 4) {
507:                 outLevel = 7;
508:             }
509:             else if (userDynamicInfo[player].levelReferAmount[5] >= 4) {
510:                 outLevel = 6;
511:             } else if (userDynamicInfo[player].levelReferAmount[4] >= 3) {
512:                 outLevel = 5;
513:             } else if (userDynamicInfo[player].levelReferAmount[3] >= 3) {
514:                 outLevel = 4;
515:             } else if (userDynamicInfo[player].levelReferAmount[2] >= 2) {
516:                 outLevel = 3;
517:             } else if (userDynamicInfo[player].levelReferAmount[1] >= 2) {
518:                 outLevel = 2;
519:             }
520:         }
521:         return outLevel;
522:     }
```

Description

biakia : The function `countingUserLevel` will read the variable `userDynamicInfo[player]` multiple times. Since the variable `userDynamicInfo[player]` is a storage struct, there is an extra sload operation which will cost 100 addition gas for each read operation. It is better to cache it as a memory variable to save gas.

Recommendation

biakia : Consider caching `userDynamicInfo[player]` as a local memory variable to save gas:

```
function countingUserLevel(address player) public view returns (uint){
        UserInfo storage info = userInfo[player];
        uint referAmount = info.referAmount;
        uint refer_R = info.refer_R;
        UserDynamicInfo memory player_dynamic = userDynamicInfo[player];
        uint level = player_dynamic.level;
        uint outLevel;
        if (levelSet[player] > 0) {
            return levelSet[player];
        }
        if (level < 1) {
            if (referAmount >= 30000 ether && refer_R >= 5) {
                outLevel = 1;
            }
        }
        if (outLevel >= 1 || level >= 1) {
            if (player_dynamic.levelReferAmount[6] >= 4) {
                outLevel = 7;
            }
            else if (player_dynamic.levelReferAmount[5] >= 4) {
                outLevel = 6;
            } else if (player_dynamic.levelReferAmount[4] >= 3) {
                outLevel = 5;
            } else if (player_dynamic.levelReferAmount[3] >= 3) {
                outLevel = 4;
            } else if (player_dynamic.levelReferAmount[2] >= 2) {
                outLevel = 3;
            } else if (player_dynamic.levelReferAmount[1] >= 2) {
                outLevel = 2;
            }
        }
        return outLevel;
    }
```

Client Response

Fixed.According to the recommendation , the code has been modified.

QTM-22:Gas Optimization: Unnecessary `pairFor` function in `CnnLibrary`::`getReserves()`

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	Hupixiong3

Code Reference

- code/router.sol#L350

```
350:         pairFor(factory, tokenA, tokenB);
```

Description

Hupixiong3 : There is an unnecessary pairFor function in the getReserves function.

Recommendation

Hupixiong3 : Delete the pairFor function.

Client Response

Fixed.According to the recommendation , we removed `pair for` function.

QTM-23:Gas Optimization:Cache array length outside for loop

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	Hupixiong3, Xi_Zi

Code Reference

- code/QEMO.sol#L50-L54
- code/stake.sol#L287-L291

```
50:    function setW(address[] memory addrs, bool b) external onlyOwner {
51:        for (uint i = 0; i < addrs.length; i++) {
52:            W[addrs[i]] = b;
53:        }
54:    }

287:    function setIDOSuper(address[] memory addrs, bool isAdd) external onlyOwner {
288:        for (uint i = 0; i < addrs.length; i++) {
289:            userIDO[addrs[i]].isSuper = isAdd;
290:        }
291:    }
```

Description

Hupixiong3 : The function setW can be optimized for gas.
Xi_Zi : In the `setIDOSuper` function of the `stake.sol` contract, located at lines 287-291, it is recommended to store the length of the `addrs` array in a separate variable outside the loop. This modification can help optimize gas consumption by avoiding repeated length calculations within the loop.

```
function setIDOSuper(address[] memory addrs, bool isAdd) external onlyOwner {
        for (uint i = 0; i < addrs.length; i++) {
            userIDO[addrs[i]].isSuper = isAdd;
        }
    }
```

Recommendation

Hupixiong3 : Gas optimization can be achieved by prioritizing the use of calldata, ++i, unchecked, and other techniques. Consider below fix in the `QEMO.setW()` function

```
    function setW(address[] calldata addrs, bool b) external onlyOwner {
        uint len = addrs.length;
        for (uint i = 0; i < len; ) {
            W[addrs[i]] = b;
            unchecked{++i;}
        }
    }
```

Xi_Zi : To optimize gas consumption, it is suggested to store the length of the `addrs` array in a separate variable before the loop. This can be done as follows:

```
uint length = addrs.length;
for (uint i = 0; i < length; i++) {
    userIDO[addrs[i]].isSuper = isAdd;
}
```

By doing so, the length of the `addrs` array is calculated only once, leading to improved gas efficiency during the loop execution.

Client Response

Fixed.According to the recommendation , the code has been modified.

QTM-24:Gas Optimization:Redundant check

Category	Severity	Status	Contributor
Gas Optimization	Informational	Declined	biakia

Code Reference

- code/stake.sol#L303-L304

```
303:    function _stake(address player, uint amount) internal {
304:        require(amount > 0, "amount is zero");
```

Description

biakia : The function `_stake` is called by `stake` and `stakeFor`. It will check whether the `amount` is zero:

```
function stake(uint amount) external checkStart checkRate onlyEOA {
        require(amount >= 100 ether, "amount is too small");
        require(userInfo[msg.sender].invitor != address(0), "not bond");
        _stake(msg.sender, amount);
    }

  function stakeFor(address player, uint amount) external checkStart checkRate {
      require(msg.sender == address(router), "not router");
      //        require(amount >= 100 ether, "amount is too small");
      if (userInfo[player].invitor == address(0)) {
          userInfo[player].invitor = address(this);
          emit Bond(player, address(this));
      }
      _stake(player, amount);
  }

function _stake(address player, uint amount) internal {
        require(amount > 0, "amount is zero");
        ...
        ...
```

However, in function `stake`, there is already check for the input param `amount`:

```
require(amount >= 100 ether, "amount is too small");
```

It is better to move the check from `_stake` to `stakeFor` to save gas.

Recommendation

biakia : Consider moving the check from `_stake` to `stakeFor` to save gas.

```
function stakeFor(address player, uint amount) external checkStart checkRate {
        require(msg.sender == address(router), "not router");
        require(amount >= 0, "amount is zero");
        if (userInfo[player].invitor == address(0)) {
            userInfo[player].invitor = address(this);
            emit Bond(player, address(this));
        }
        _stake(player, amount);
    }

function _stake(address player, uint amount) internal {
        // require(amount > 0, "amount is zero");
        require(usdt.balanceOf(player) >= amount, "usdt balance not enough");
        usdt.transferFrom(msg.sender, address(this), amount);
        ...
         ...
```

Client Response

Declined. `stakeFor` does not need for verification, but the `stake` requires verification.

QTM-25:Gas Optimization:Unused Function

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	Xi_Zi

Code Reference

- code/router.sol#L599-L610

```
599:    function _swap(uint[] memory amounts, address[] memory path, address _to) internal virtual {
600:        for (uint i; i < path.length - 1; i++) {
601:            (address input, address output) = (path[i], path[i + 1]);
602:            (address token0,) = CnnLibrary.sortTokens(input, output);
603:            uint amountOut = amounts[i + 1];
604:            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOut) : (amoun
tOut, uint(0));
605:            address to = i < path.length - 2 ? CnnLibrary.pairFor(factory, output, path[i + 2])
: _to;
606:            ICnnPair(CnnLibrary.pairFor(factory, input, output)).swap(
607:                amount0Out, amount1Out, to, new bytes(0)
608:            );
609:        }
610:    }
```

Description

Xi_Zi : The `_swap` function in the `router.sol` contract, located at lines 599-610, is defined but not used within the contract. This indicates that the function is not being called or referenced anywhere in the contract's code.

```
function _swap(uint[] memory amounts, address[] memory path, address _to) internal virtual {
    // Function implementation
}
```

Recommendation

Xi_Zi : If the `_swap` function is not intended to be used in the current implementation, it can be safely removed to improve code readability and maintenance. However, if there are plans to utilize this function in the future, ensure that it is properly called or referenced in the contract's code.

Client Response

Fixed.Deleted the `_swap` method

QTM-26:Gas Optimization:Unused functions

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	biakia, Hupixiong3

Code Reference

- code/stake.sol#L129-L131
- code/my_factory.sol#L478-L482
- code/router.sol#L599-L610

```
129:    function formatTime(uint timestamp) internal pure returns (uint) {
130:        return timestamp - (timestamp - 16 * 3600) % 86400;
131:    }

478:    function getBurnAmount() internal view returns (uint){
479:        uint balance0 = getToken0Blance();
480:        uint _burnAmount = balance0 - reserve0;
481:        return _burnAmount;
482:    }

599:    function _swap(uint[] memory amounts, address[] memory path, address _to) internal virtual {
600:        for (uint i; i < path.length - 1; i++) {
601:            (address input, address output) = (path[i], path[i + 1]);
602:            (address token0,) = CnnLibrary.sortTokens(input, output);
603:            uint amountOut = amounts[i + 1];
604:            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOut) : (amoun
tOut, uint(0));
605:            address to = i < path.length - 2 ? CnnLibrary.pairFor(factory, output, path[i + 2])
: _to;
606:            ICnnPair(CnnLibrary.pairFor(factory, input, output)).swap(
607:                amount0Out, amount1Out, to, new bytes(0)
608:            );
609:        }
610:    }
```

Description

biakia : In contract `CnnRouter`, the internal function `_swap` is never used.
biakia : In contract `QEMOStake`, the internal function `formatTime` is never used.

Hupixiong3 : The function "getBurnAmount" is not used and is considered a redundant function.

Recommendation

biakia : If the internal function is not intended to be used, it is better to remove this function to save gas.
biakia : In the internal function is not intended to be used, it is better to remove this function to save gas.
Hupixiong3 : If the "getBurnAmount" function is not being used and is considered redundant, you can safely delete it from your code. However, if you still need the functionality provided by that function, you can modify it or integrate its features elsewhere in your code to ensure it serves a useful purpose.

Client Response

Fixed.According to the recommendation , the code has been modified.

QTM-27:Gas Optimization:Unused variable

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	infinityhacker, biakia, Hupixiong3, Xi_Zi

Code Reference

- code/stake.sol#L12
- code/QEMO.sol#L17
- code/stake.sol#L55
- code/my_factory.sol#L310
- code/my_factory.sol#L526
- code/my_factory.sol#L526-L528
- code/my_factory.sol#L565

```
12:    uint public left;

17:    uint public transferFee;

55:    mapping(address => bool) public admin;

310:    uint public burnAmount;

526:            if (data.length > 0) {

526:            if (data.length > 0) {
527:
528:            }

565:    bytes32 public constant INIT_CODE_PAIR_HASH = keccak256(abi.encodePacked(type(QEMOPair).creationCode));
```

Description

infinityhacker : Currently the `data` field in swap function does not use, and the pool seems does not support flashloan currently, so it's recommend to be removed
biakia : In the contract `QEMOPair`, the variable `burnAmount` is neither initialized nor updated, which means the value of `burnAmount` is always 0.
biakia : In the contract `QEMOStake`, the variable `left` and `admin` are never used.
biakia : In contract `Quantemo`, the variable `transferFee` is never used.
Hupixiong3 : There is a conditional statement data.length > 0 in the swap function, but the corresponding scope is

empty.

Hupixiong3 : The constant INIT_CODE_PAIR_HASH is not used.

Xi_Zi : In the file `code/my_factory.sol`, lines 526-528 contain unused code that serves no purpose. These lines appear within the `swap` function and do not contribute to the functionality or logic of the contract.

```
function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
        ...
            // optimistically transfer tokens
            if (data.length > 0) {

            }//@audit unused code

        ...
    }
```

Recommendation

infinityhacker : Remove the `data` field

biakia : If the variable `burnAmount` is not intended to be used, it is recommended to remove this variable to save gas.

biakia : If these variables are not intended to be used, it is recommended to remove them to save gas.

biakia : If the variable `transferFee` is not intended to be used, it is better to remove this variable to save gas.

Hupixiong3 : Remove the conditional statement.

Hupixiong3 : Delete the constant INIT_CODE_PAIR_HASH.

Xi_Zi : Remove the unused code from lines 526-528 within the `swap` function in the `my_factory.sol` file to improve code readability and maintainability. Unused code adds unnecessary complexity and can confuse developers who review or work with the contract.

Client Response

Fixed.According to the recommendation , the code has been modified.

QTM-28:Gas Optimization:Use `calldata` instead of `memory` as much as possible

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	biakia, Xi_Zi

Code Reference

- code/QEMO.sol#L50-L54
- code/stake.sol#L287-L291

```
50:    function setW(address[] memory addrs, bool b) external onlyOwner {
51:        for (uint i = 0; i < addrs.length; i++) {
52:            W[addrs[i]] = b;
53:        }
54:    }

287:    function setIDOSuper(address[] memory addrs, bool isAdd) external onlyOwner {
288:        for (uint i = 0; i < addrs.length; i++) {
289:            userIDO[addrs[i]].isSuper = isAdd;
290:        }
291:    }
```

Description

biakia : It's better to use `calldata` instead of `memory` for function parameters that represent variables that will not be modified.
Xi_Zi : In the `setIDOSuper` function of the `stake.sol` contract, it is recommended to use the `calldata` keyword in the function parameter `addrs` to optimize gas consumption. Since the function only reads the values from `addrs` and does not modify them, using `calldata` instead of `memory` can save gas costs.

```
function setIDOSuper(address[] memory addrs, bool isAdd) external onlyOwner {
    for (uint i = 0; i < addrs.length; i++) {
        userIDO[addrs[i]].isSuper = isAdd;
    }
}
```

Recommendation

biakia : Consider using calldata instead of memory to save gas.

Xi_Zi : Update the `setIDOSuper` function by changing the function parameter `addrs` from `memory` to `calldata`:

- From: `address[] memory addrs`
- To: `address[] calldata addrs`

This change optimizes gas consumption by indicating that the function parameter will only be read and not modified.

Client Response

Fixed.According to the recommendation , the code has been modified.

QTM-29:Gas Optimization:Use immutable as much as possible

Category	Severity	Status	Contributor
Language Specific	Informational	Acknowledged	biakia, Hupixiong3

Code Reference

- code/QEMO.sol#L17-L19
- code/QEMO.sol#L23-L31
- code/router.sol#L457

```
17:    uint public transferFee;
18:    uint public pancakeFee;
19:    uint public insideFee;

23:    constructor() ERC20("Quantemo", "QEMO") {
24:        _mint(msg.sender, 9900000 ether);
25:        freeLimit = 9800000 ether;
26:        transferFee = 1;
27:        pancakeFee = 2;
28:        insideFee = 2;
29:        balanceLimit = 1e13;
30:
31:    }

457:    address public QEMO;
```

Description

biakia :

```
    uint public transferFee;
    uint public pancakeFee;
    uint public insideFee;
```

The linked variables assigned in the constructor can be declared as immutable. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Hupixiong3 : QEMO is set through the constructor function and cannot be modified by any other functions.

Recommendation

biakia : We recommend declaring these variables as immutable. Please note that the immutable keyword only works in Solidity version v0.6.5 and up.

```
    uint public immutable transferFee;
    uint public immutable pancakeFee;
    uint public immutable insideFee;
```

Hupixiong3 : Use the immutable modifier to designate QEMO as an immutable variable.

Client Response

Acknowledged.

QTM-30:Missing event record

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	Xi_Zi

Code Reference

- code/stake.sol#L8
- code/QEMO.sol#L9

```
8:contract QEMOStake is OwnableUpgradeable {

9:contract Quantemo is ERC20, Ownable {
```

Description

Xi_Zi : The owner's privileged operations in the `QEMO` contract and `stake` contract are missing corresponding events. Events are crucial for transparency and allowing external systems to track changes in the contract state, especially when it involves privileged actions.

Recommendation

Xi_Zi : Add appropriate events for owner's privileged operations in the `QEMO` contract and `stake` contract. Emitting events with relevant information will provide visibility into these actions and enable better monitoring of the contract.

Examples：

```
event AdminUpdated(address indexed admin, bool isAdmin);

function setAdmin(address admin_, bool isAdmin_) external onlyOwner {
    admin[admin_] = isAdmin_;
    emit AdminUpdated(admin_, isAdmin_);
}
```

Client Response

Fixed. According to recommendation,we add event record.

QTM-31:Unlocked Pragma Version

Category	Severity	Status	Contributor
Language Specific	Informational	Fixed	biakia

Code Reference

- code/QEMO.sol#L2
- code/dynamic.sol#L2
- code/stake.sol#L2
- code/router.sol#L5

```
2:pragma solidity ^0.8.0;

2:pragma solidity ^0.8.0;

2:pragma solidity ^0.8.0;

5:pragma solidity >=0.6.0;
```

Description

biakia : Solidity files in packages have a pragma version `^0.8.0`. The caret (^) points to unlocked pragma, meaning the compiler will use the specified version or above.

Recommendation

biakia : It's good practice to use specific solidity versions to know compiler bug fixes and optimisations were enabled at the time of compiling the contracts.

Client Response

Fixed. According to recommendation,we locked pragma version.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.