Smart Contract Security Audit Report

For

W3BANK

Date Issued: Aug 17, 2023

Version: v1.0

Confidentiality Level: Public

Contents

1 Abstract

This report was prepared for W3BANK smart contract to identify issues and vulnerabilities in its smart contract source code. A thorough examination of W3BANK smart contracts was conducted through timely communication with W3BANK, static analysis using multiple audit tools and manual auditing of their smart contract source code.

The audit process paid particular attention to the following considerations.

- A thorough review of the smart contract logic flow
- Assessment of the code base to ensure compliance with current best practice and industry standards
- Ensured the contract logic met the client's specifications and intent
- Internal vulnerability scanning tools tested for common risks and writing errors
- Testing smart contracts for common attack vectors
- Test smart contracts for known vulnerability risks
- Conduct a thorough line-by-line manual review of the entire code base

As a result of the security assessment, issues ranging from critical to informational were identified. We recommend that these issues are addressed to ensure a high level of security standards and industry practice. The recommendations we made could have better served the project from a security perspective.

- Enhance general coding practices to improve the structure of the source code.
- Provide more comments for each function to improve readability.
- Provide more transparency of privileged activities once the agreement is in place.

2 Overview

2.1 Project Summary

Project Summary	Project Information
Name	W3BANK
Start date	Aug 10, 2023
End date	Aug 17, 2023
Platform	PEGO Network
Contract type	DeFi
Language	Solidity
Audit content	https://github.com/pego-labs/w3bank-lending-feed
Commits	4f41e2b5079f29f6581befa670b90c54000919f2
File	Achievement.sol, AssistedReading.sol、InterestRateModel.sol、LendController.sol、LendTokenFactory.sol、Migrations.sol、OracelBridge.sol、OracleFactory.sol、TokenTemplate.sol、OracleTemplate.sol

2.2 Report HASH

Name	HASH
W3BANK	7A33881021B25CE39330FF8E05AD0C6CAB7ABEA6E42FBC02647925184B7AFABC

3 Project contract details

3.1 Contract Overview

OracleTemplate.sol

The contract is a fixed-window oracle contract, which is used to provide price information of trading pairs for other smart contracts to use. It uses the Uniswap V2 interface contract and library contract to obtain the price of the transaction pair, and calculates the average price. Judging by the implementation mechanism of the contract, the contract administrator will periodically call the specified function to update the price and calculate the average value to provide more stable price data.

AssistedReading.sol

Auxiliary query contract, mainly used for front-end query data.

InterestRateModel.sol

The contract mainly implements the interest rate algorithm required in the project, and calculates the lending rate and supply rate based on the fund usage rate and other parameters.

LendController.sol

The contract implements the management and control functions of the lending market, including adding markets, setting parameters, calculating users' available funds, and calculating liquidation rewards, etc. At the same time, it also provides an interface for users to participate in the market, withdraw from the market and extract platform benefits.

LendTokenFactory.sol

This contract implements a factory contract for deploying the market contract based on the TokenTemplate contract. It also records all market contracts created through this contract; and all users can query.

Migrations.sol

Migration contract, used to record the migration history and version information of the contract.

OracelBridge.sol

The function of this contract is to serve as the general oracle of the lending market, bridging the actual price oracles associated with different tokens. By setting the price oracle address of different loan certificates and burning tokens, the actual price of the underlying asset can be obtained.

OracleFactory.sol

This contract implements the oracle machine construction contract, which allows the creation and management of multiple oracle machines. The contract saves all the oracle machines created through this contract, and provides the operation of batch updating prices for all oracle machines.

TokenTemplate.sol

The contract implements a loan market contract template. Users can obtain loan certificates by depositing assets, lend assets and repay the loan. Users can redeem deposited assets, and other users can liquidate loans and obtain collateral. The manager of the contract can set the borrowing parameters and the receiving address of the liquidation reward, etc. The contract provides basic lending market functions.

3.2 Code Overview

OracleTemplate Contract

Function Name	Visibility	Modifiers
initialize	External	initializer
checkUpdate	External	-
update	External	-
consult	External	-

AssistedReading Contract

Function Name	Visibility	Modifiers
initialize	External	initializer
getCollateralFactor	Internal	-
getMarketDetailOne	Public	-
getMarketDetail	External	-
getDebtor	External	-
getBorrowBalance	External	-
bestLiquidation	External	-

InterestRateModel Contract

Function Name	Visibility	Modifiers
initialize	External	initializer
utilizationRate	Public	-
getBorrowRate	Public	-
getSupplyRate	Public	-

Achievement Contract

Function Name	Visibility	Modifiers
initialize	External	initializer
setFactory	External	-
addMarket	External	onlyRole(MANAGER_ROLE)
setCollateralFactorr	External	onlyRole(MANAGER_ROLE)
setBorrowCaps	External	onlyRole(MANAGER_ROLE)
setMintSwitch	External	onlyRole(MANAGER_ROLE)
setBorrowSwitch	External	onlyRole(MANAGER_ROLE)
setFarmOutPutPerBlock	External	onlyRole(MANAGER_ROLE)
setPoolPoint	External	onlyRole(MANAGER_ROLE)
setPriceOracle	External	onlyRole(DEFAULT_ADMIN_ROLE)
setBurnToken	External	onlyRole(MANAGER_ROLE)
getBurnAmount	External	checkListed
beforeTransfer	External	checkListed
beforeMint	External	checkListed
beforeBorrow	External	checkListed
beforeRepayBorrow	External	checkListed
beforeRedeem	External	checkListed
beforeSeize	External	checkListed
getMaxRedeem	External	-
liquidateCalculateSeizeTokens	External	-
checkLiquidateBorrow	External	-
getHypotheticalAccountLiquidity	Public	-
checkMembership	External	-

enterMarkets	External	-
exitMarket	External	-
addToMarket	Internal	-
updatePool	Public	-
takeReward	External	-
takeRewardFromMarket	Internal	-
deposit	Internal	-
withdraw	Internal	-
earned	External	-
earnedByMarket	Internal	-
getTokenPerShare	Internal	-
getAllMarkets	External	-
getAssetsIn	External	-
isDeprecated	Public	-

OracleFactory Contract

Function Name	Visibility	Modifiers
initialize	External	initializer
getAllLOracle	External	-
checkUpdate	External	-
update	External	onlyRole(MANAGER_ROLE)
createOracle	External	-

TokenTemplate Contract

Function Name	Visibility	Modifiers
initialize	External	-
accrueInterestPublic	Public	-
_updateBorrowSnapshot	Internal	-
setBorrowRateByPerid	External	-
setadminSeizeReceiver	External	-
_beforeTokenTransfer	Internal	-
mint	External	-
borrow	External	-
repayBorrow	External	-
_repayBorrow	Internal	-
redeem	External	-
liquidateBorrow	External	-
seize	External	-
_seize	Internal	-
getCurrentPeriod	Public	-
updateBorrowByPeriod	Internal	-
exchangeRateCurrent	Public	-
getCash	Public	-
balanceOfUnderlying	External	-
borrowBalance	Public	-
getAccountSnapshot	External	-
supplyRatePerBlock	External	-
borrowRatePerBlock	External	-

utilizationRate	External	-
doTransferIn	Internal	-
doTransferOut	Internal	-

LendTokenFactory Contract

Function Name	Visibility	Modifiers
initialize	External	initializer
getAllLtokens	External	-
createLendToken	External	onlyRole(MANAGER_ROLE)

OracelBridge Contract

Function Name	Visibility	Modifiers
initialize	External	initializer
setBurnToken	External	onlyRole(DEFAULT_ADMIN_ROLE)
setPriceOracle	External	onlyRole(DELEGATE_ROLE)
getUnderlyingPrice	External	-

4 Audit results

4.1 Key messages

ID	Title	Severity	Status
01	timeElapsed >= PERIOD condition repeated	Informational	confirmed
02	factory can be modified by any caller	Medium	fixed
03	The collateralFactorMantissa may be zero when the market is updated	Informational	confirmed
04	Setting the mining pool coefficient may appear to be zero	Informational	confirmed
05	Nonsensical variable judgment condition	Informational	confirmed
06	isBorrowOpenedOf[msg.sender] judgment condition is repeated	Informational	confirmed
07	Funds transfer sequence is not secure	Medium	fixed
08	Privileged roles can update contract variables	Low	confirmed

4.2 Audit details

4.2.1 timeElapsed >= PERIOD condition repeated

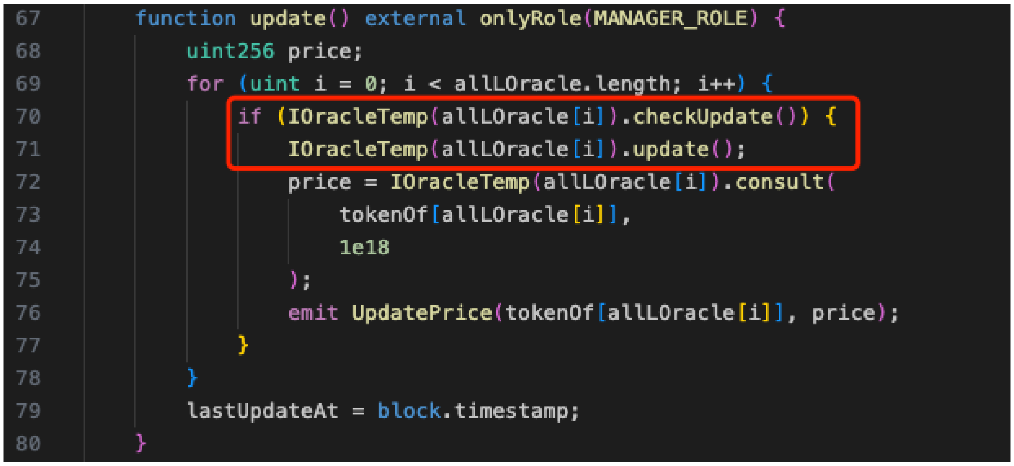
ID	Severity	Location	Status
01	Informational	OracleTemplate.sol: 54, 86	confirmed

Description

Both checkUpdate() and update() in the OracleTemplate contract will judge whether the condition of timeElapsed >= PERIOD is satisfied. The above two methods are called by the update() method of the OracleFactory contract. The calling order of the method is to call OracleTemplate.checkUpdate() first and then OracleTemplate. update(). Since checkUpdate() has already judged the timeElapsed >= PERIOD condition, and OracleTemplate.update() will not be executed when the condition is not met, but if the condition is met, the conditions in both methods will be met.

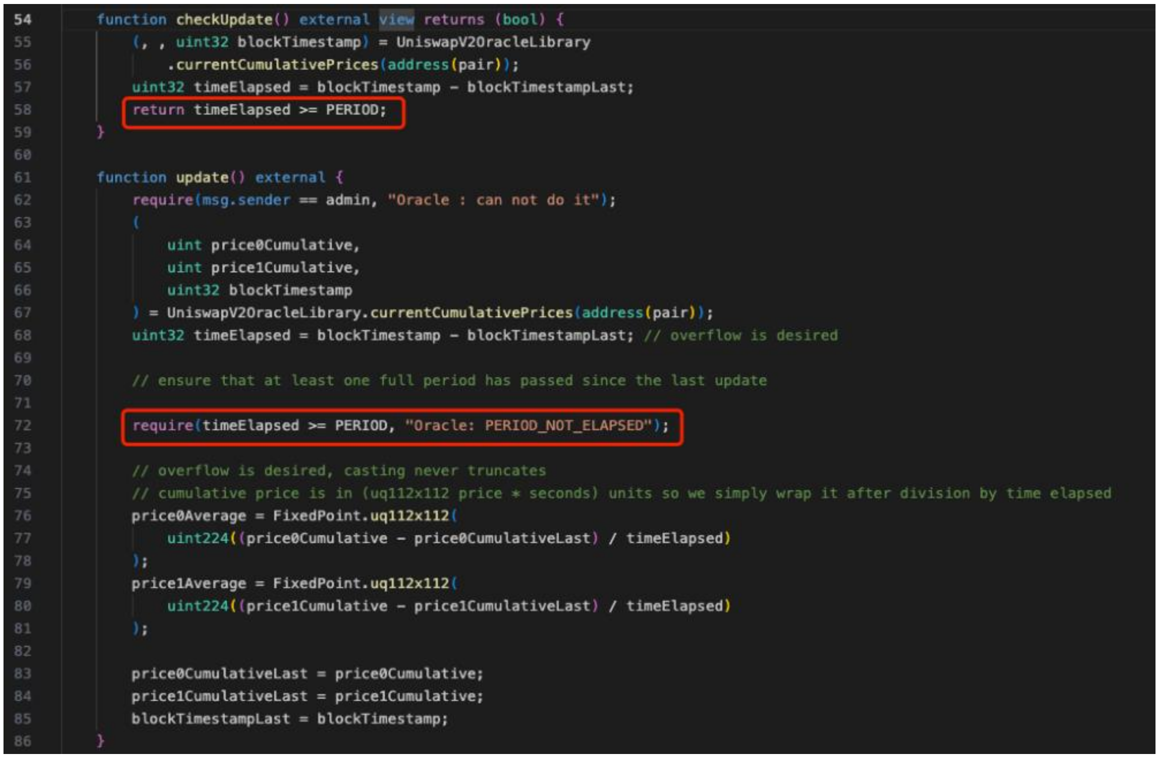
Code location:

OracleFactory.sol



```
    function update() external onlyRole(MANAGER_ROLE) {
        uint256 price;
        for (uint i = 0; i < allLOracle.length; i++) {
            if (IOracleTemp(allLOracle[i]).checkUpdate()) {
                IOracleTemp(allLOracle[i]).update();
                price = IOracleTemp(allLOracle[i]).consult(
                    tokenOf[allLOracle[i]],
                    1e18
                );
                emit UpdatePrice(tokenOf[allLOracle[i]], price);
            }
        }
        lastUpdateAt = block.timestamp;
    }
```

OracleTemplate.sol



```
function checkUpdate() external view returns (bool) {
    (, , uint32 blockTimestamp) = UniswapV2OracleLibrary
        .currentCumulativePrices(address(pair));
    uint32 timeElapsed = blockTimestamp - blockTimestampLast;
    return timeElapsed >= PERIOD;
}

function update() external {
    require(msg.sender == admin, "Oracle : can not do it");
    (
        uint price0Cumulative,
        uint price1Cumulative,
        uint32 blockTimestamp
    ) = UniswapV2OracleLibrary.currentCumulativePrices(address(pair));
    uint32 timeElapsed = blockTimestamp - blockTimestampLast; // overflow is desired

    // ensure that at least one full period has passed since the last update

    require(timeElapsed >= PERIOD, "Oracle: PERIOD_NOT_ELAPSED");

    // overflow is desired, casting never truncates
    // cumulative price is in (uq112x112 price * seconds) units so we simply wrap it after division by time elapsed
    price0Average = FixedPoint.uq112x112(
        uint224((price0Cumulative - price0CumulativeLast) / timeElapsed)
    );
    price1Average = FixedPoint.uq112x112(
        uint224((price1Cumulative - price1CumulativeLast) / timeElapsed)
    );

    price0CumulativeLast = price0Cumulative;
    price1CumulativeLast = price1Cumulative;
    blockTimestampLast = blockTimestamp;
}
```

Recommendation

It is recommended to delete the timeElapsed >= PERIOD condition judgment in the OracleTemplate.update() method.

Status

confirmed.

The project party responded that the oracle instance contract is not necessarily created and managed by the oracleFactory contract, so this redundant judgment is necessary.

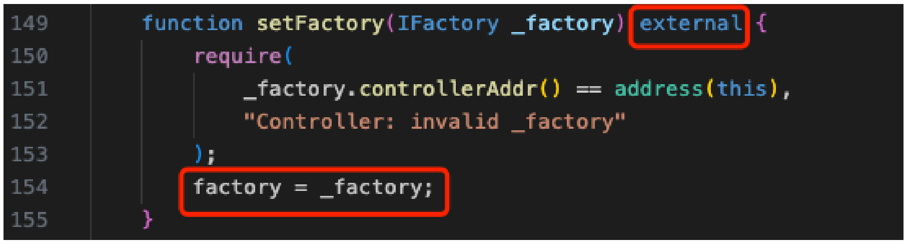
4.2.2 factory can be modified by any caller

ID	Severity	Location	Status
02	Medium	LendController.sol: 149, 155	fixed

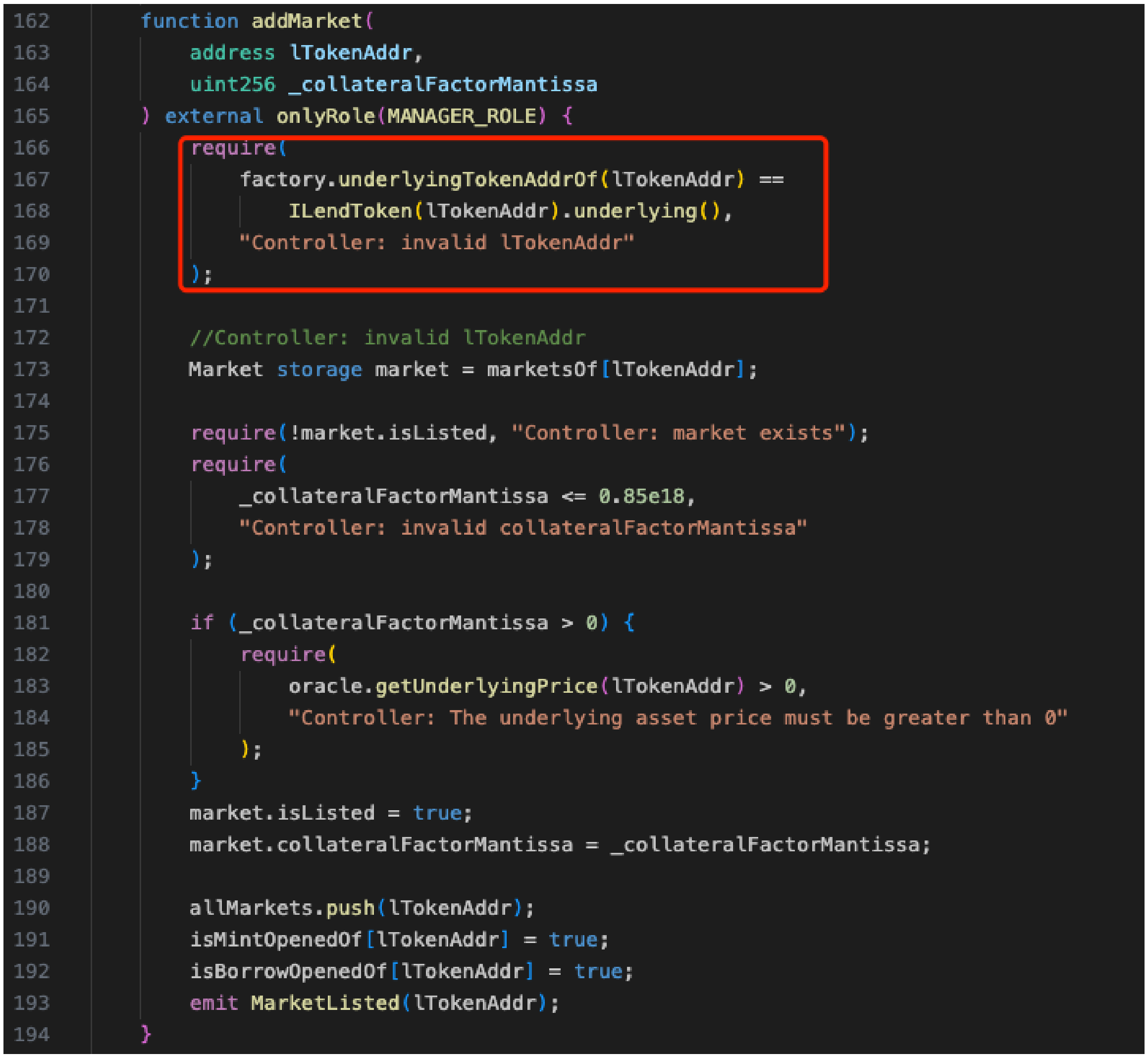
Description

The setFactory() method is used to set the factory. The attacker can make the return value of the _factory contract controllerAddr() method equal to address(this) by constructing the _factory contract, and finally complete the modification of the factory. The factory mainly makes judgments when adding a market. When the return value of underlyingTokenAddrOf is maliciously set when adding a market, the judgment condition will fail. Of course, you can continue to call the factory to make changes here.

Code location:



```
149     function setFactory(IFactory _factory) external {
150         require(
151             _factory.controllerAddr() == address(this),
152             "Controller: invalid _factory"
153         );
154         factory = _factory;
155     }
```



```
    function addMarket(
        address lTokenAddr,
        uint256 _collateralFactorMantissa
    ) external onlyRole(MANAGER_ROLE) {
        require(
            factory.underlyingTokenAddrOf(lTokenAddr) ==
                ILendToken(lTokenAddr).underlying(),
            "Controller: invalid lTokenAddr"
        );

        //Controller: invalid lTokenAddr
        Market storage market = marketsOf[lTokenAddr];

        require(!market.isListed, "Controller: market exists");
        require(
            _collateralFactorMantissa <= 0.85e18,
            "Controller: invalid collateralFactorMantissa"
        );

        if (_collateralFactorMantissa > 0) {
            require(
                oracle.getUnderlyingPrice(lTokenAddr) > 0,
                "Controller: The underlying asset price must be greater than 0"
            );
        }
        market.isListed = true;
        market.collateralFactorMantissa = _collateralFactorMantissa;

        allMarkets.push(lTokenAddr);
        isMintOpenedOf[lTokenAddr] = true;
        isBorrowOpenedOf[lTokenAddr] = true;
        emit MarketListed(lTokenAddr);
    }
```

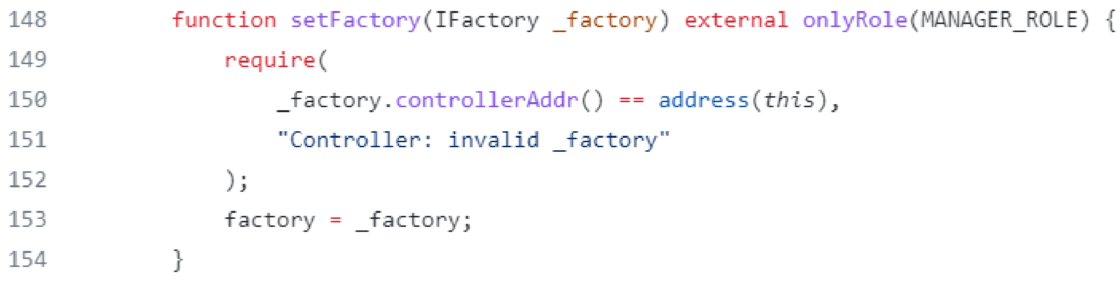
Recommendation

It is recommended to restrict callers of this method to administrators.

Status

fixed.

commits:f16195d1708db62fbe208cdec83b3bdb3ea45b11



```
    function setFactory(IFactory _factory) external onlyRole(MANAGER_ROLE) {
        require(
            _factory.controllerAddr() == address(this),
            "Controller: invalid _factory"
        );
        factory = _factory;
    }
```

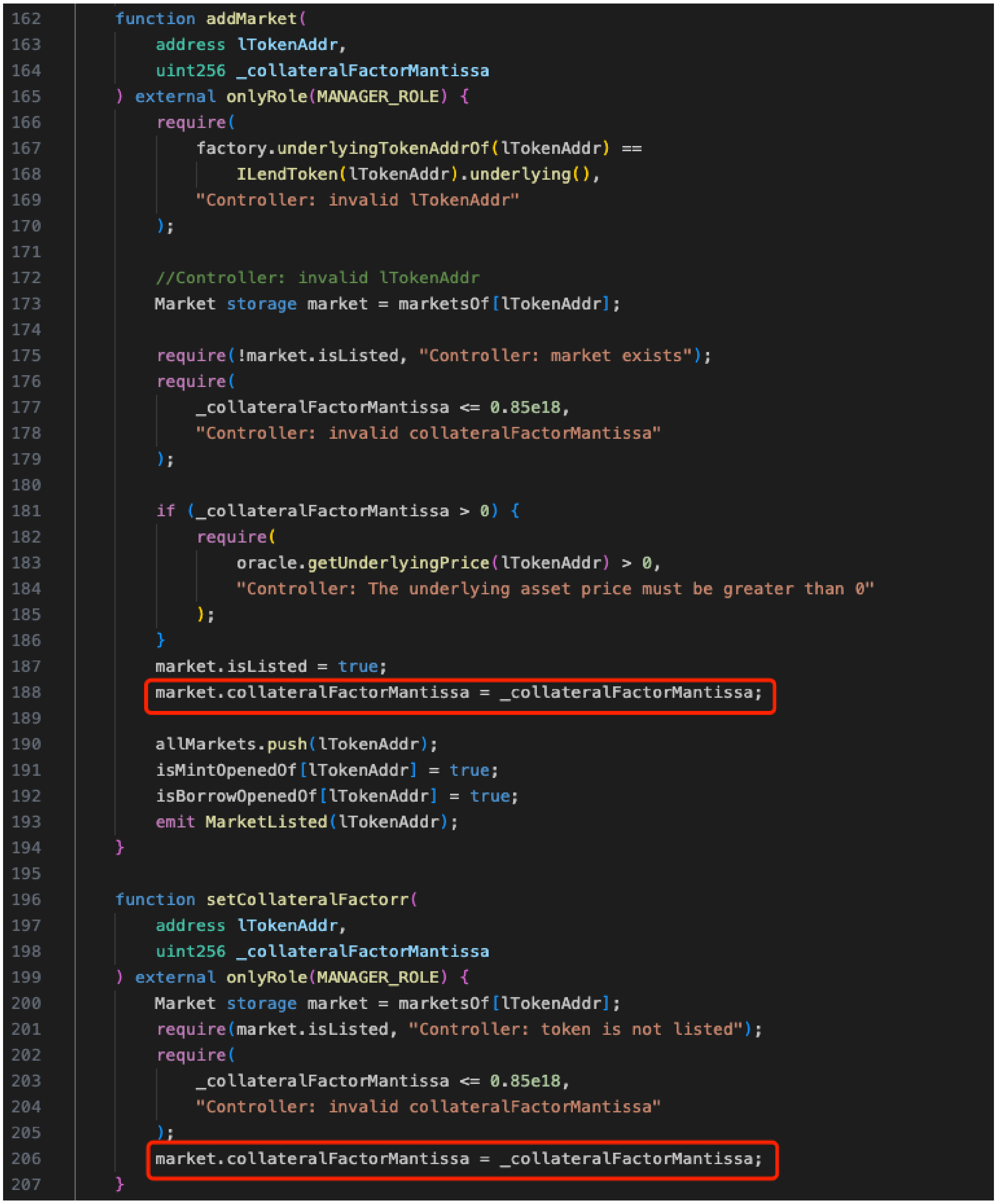
4.2.3 The collateralFactorMantissa may be zero when the market is updated

ID	Severity	Location	Status
03	Informational	LendController.sol: 162, 207	confirmed

Description

When addMarket() and setCollateralFactorr() update collateralFactorMantissa, it is judged that _collateralFactorMantissa cannot be greater than 0.85e18, but there is no whether _collateralFactorMantissa is zero.

Code location:



```
    function addMarket(
        address lTokenAddr,
        uint256 _collateralFactorMantissa
    ) external onlyRole(MANAGER_ROLE) {
        require(
            factory.underlyingTokenAddrOf(lTokenAddr) ==
                ILendToken(lTokenAddr).underlying(),
            "Controller: invalid lTokenAddr"
        );

        //Controller: invalid lTokenAddr
        Market storage market = marketsOf[lTokenAddr];

        require(!market.isListed, "Controller: market exists");
        require(
            _collateralFactorMantissa <= 0.85e18,
            "Controller: invalid collateralFactorMantissa"
        );

        if (_collateralFactorMantissa > 0) {
            require(
                oracle.getUnderlyingPrice(lTokenAddr) > 0,
                "Controller: The underlying asset price must be greater than 0"
            );
        }
        market.isListed = true;
        market.collateralFactorMantissa = _collateralFactorMantissa;

        allMarkets.push(lTokenAddr);
        isMintOpenedOf[lTokenAddr] = true;
        isBorrowOpenedOf[lTokenAddr] = true;
        emit MarketListed(lTokenAddr);
    }

    function setCollateralFactorr(
        address lTokenAddr,
        uint256 _collateralFactorMantissa
    ) external onlyRole(MANAGER_ROLE) {
        Market storage market = marketsOf[lTokenAddr];
        require(market.isListed, "Controller: token is not listed");
        require(
            _collateralFactorMantissa <= 0.85e18,
            "Controller: invalid collateralFactorMantissa"
        );
        market.collateralFactorMantissa = _collateralFactorMantissa;
    }
```

Recommendation

It is recommended to judge that _collateralFactorMantissa cannot be zero.

Status

confirmed. The project side responded that collateralFactorMantissa is allowed to be 0 in the business. If it is 0, it means that the currency cannot be calculated as liquidity.

4.2.4 Setting the mining pool coefficient may appear to be zero

ID	Severity	Location	Status
04	Informational	LendController.sol: 268, 299	confirmed

Description

The setPoolPoint() method is used to set the mine pool coefficient. There are four situations when setting this method, which are

1) pool.allocPoint == 0 && allocPoint > 0

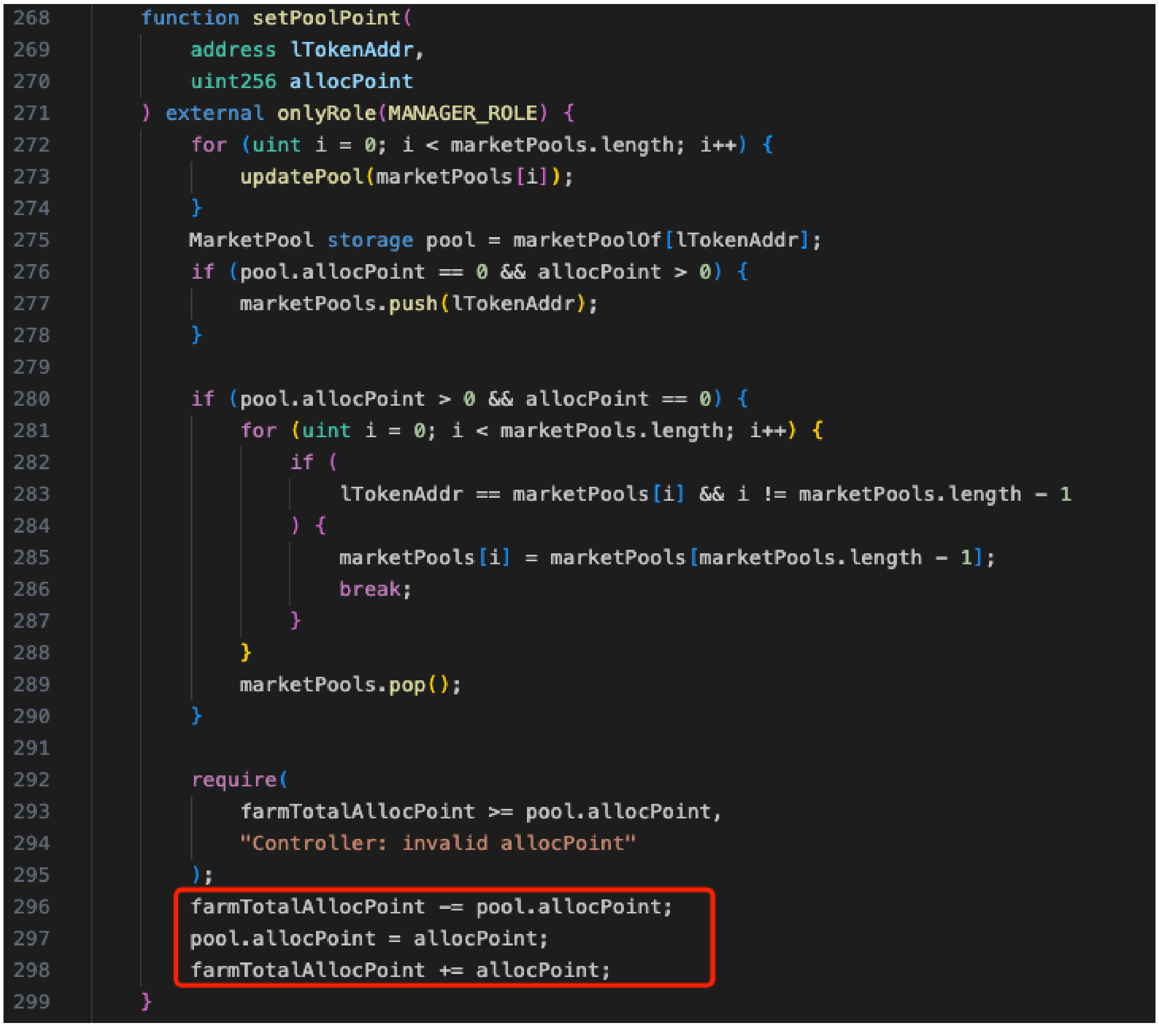
2) pool.allocPoint > 0 && allocPoint == 0

3) pool.allocPoint > 0 && allocPoint > 0

4) pool.allocPoint == 0 && allocPoint == 0

Among them, the first two have been judged in the setPoolPoint() method, which are mainly used for adding and deleting. The third case is to update the coefficient, and the fourth case can be executed normally. But it doesn't make any sense.

Code location:



```
function setPoolPoint(
    address lTokenAddr,
    uint256 allocPoint
) external onlyRole(MANAGER_ROLE) {
    for (uint i = 0; i < marketPools.length; i++) {
        updatePool(marketPools[i]);
    }
    MarketPool storage pool = marketPoolOf[lTokenAddr];
    if (pool.allocPoint == 0 && allocPoint > 0) {
        marketPools.push(lTokenAddr);
    }

    if (pool.allocPoint > 0 && allocPoint == 0) {
        for (uint i = 0; i < marketPools.length; i++) {
            if (
                lTokenAddr == marketPools[i] && i != marketPools.length - 1
            ) {
                marketPools[i] = marketPools[marketPools.length - 1];
                break;
            }
        }
        marketPools.pop();
    }

    require(
        farmTotalAllocPoint >= pool.allocPoint,
        "Controller: invalid allocPoint"
    );
    farmTotalAllocPoint -= pool.allocPoint;
    pool.allocPoint = allocPoint;
    farmTotalAllocPoint += allocPoint;
}
```

Recommendation

It is recommended to add judgment so that the condition of pool.allocPoint == 0 && allocPoint == 0 cannot be executed normally.

Status

confirmed.

The project side responded that setPoolPoint is also allowed to appear in the business. The proportion of the mining pool is 0, and 0 means that the currency does not generate income.

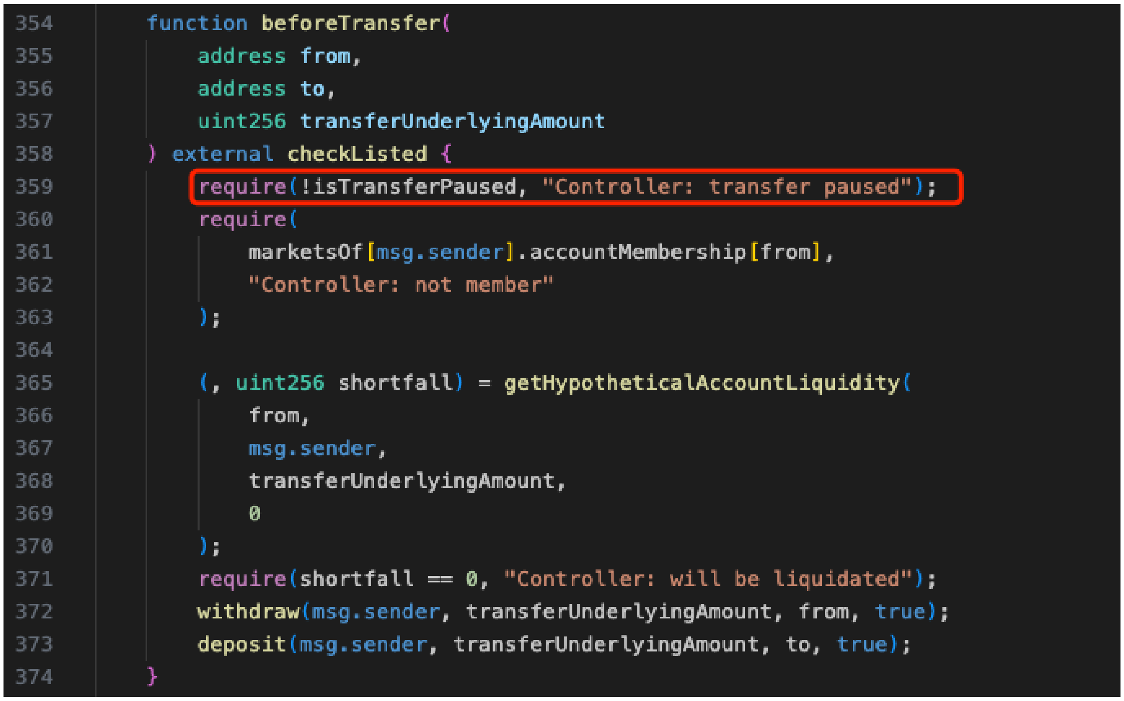
4.2.5 Nonsensical variable judgment condition

ID	Severity	Location	Status
05	Informational	LendController.sol: 354, 374; 442, 458	confirmed

Description

isTransferPaused is a global variable, the default is false, and this variable will not change, always false. In the beforeTransfer() method, the result of judging the !isTransferPaused condition will always be satisfied, and the judgment will be meaningless.

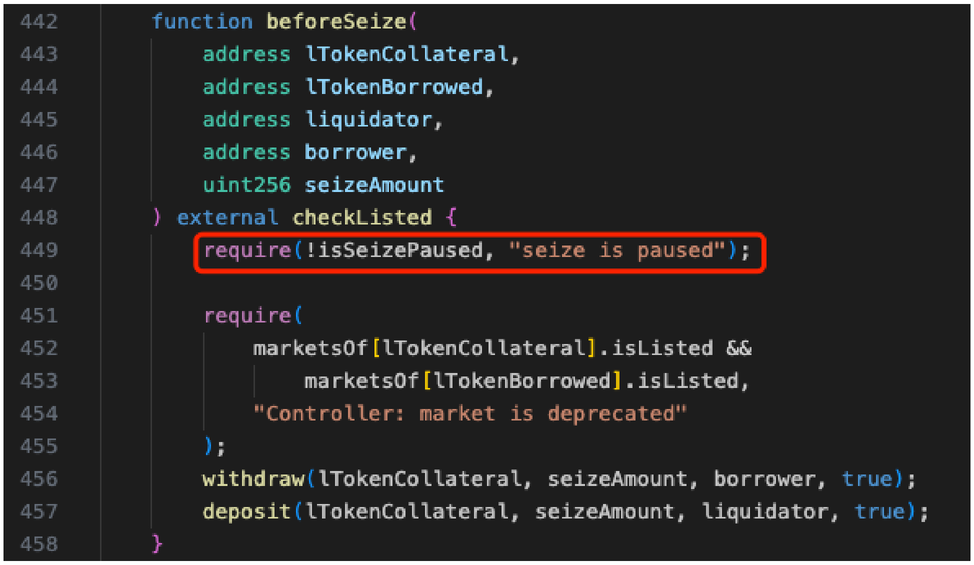
Code location:



```
354     function beforeTransfer(
355         address from,
356         address to,
357         uint256 transferUnderlyingAmount
358     ) external checkListed {
359         require(!isTransferPaused, "Controller: transfer paused");
360         require(
361             marketsOf[msg.sender].accountMembership[from],
362             "Controller: not member"
363         );
364
365         (, uint256 shortfall) = getHypotheticalAccountLiquidity(
366             from,
367             msg.sender,
368             transferUnderlyingAmount,
369             0
370         );
371         require(shortfall == 0, "Controller: will be liquidated");
372         withdraw(msg.sender, transferUnderlyingAmount, from, true);
373         deposit(msg.sender, transferUnderlyingAmount, to, true);
374     }
```

isSeizePaused is a global variable, the default is false, and this variable will not change, always false. In the beforeSeize() method, the result of judging the !isSeizePaused condition will always be satisfied, and the judgment will be meaningless.

Code location:



```
    function beforeSeize(
        address lTokenCollateral,
        address lTokenBorrowed,
        address liquidator,
        address borrower,
        uint256 seizeAmount
    ) external checkListed {
        require(!isSeizePaused, "seize is paused");

        require(
            marketsOf[lTokenCollateral].isListed &&
                marketsOf[lTokenBorrowed].isListed,
            "Controller: market is deprecated"
        );
        withdraw(lTokenCollateral, seizeAmount, borrower, true);
        deposit(lTokenCollateral, seizeAmount, liquidator, true);
    }
```

Recommendation

It is recommended to delete the meaningless judgment condition.

Status

confirmed.

The project side responds to isTransferPaused and isSeizePaused although they are static variables in the contract and cannot be modified. But the contract itself can be upgraded, and changes can be made by upgrading the contract. In order to control the behavior of the entire market when necessary.

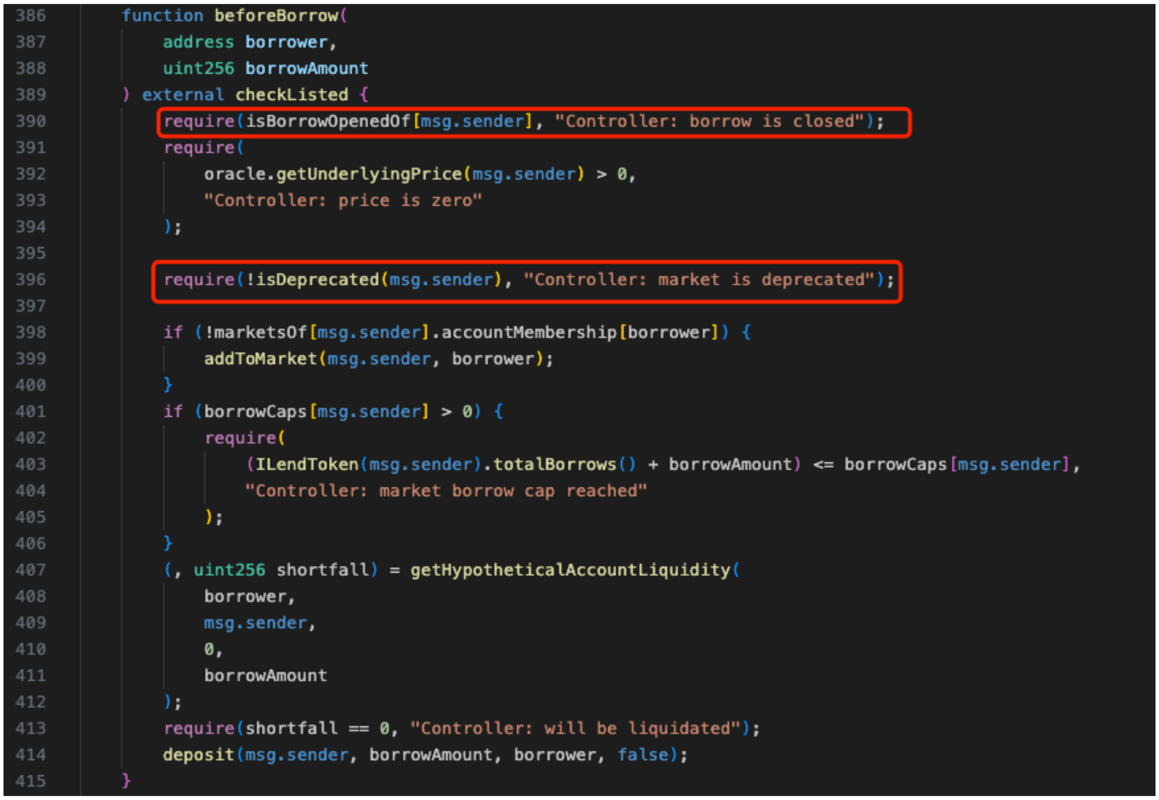
4.2.6 isBorrowOpenedOf[msg.sender] judgment condition is repeated

ID	Severity	Location	Status
06	Informational	LendController.sol: 386, 415; 888, 893	confirmed

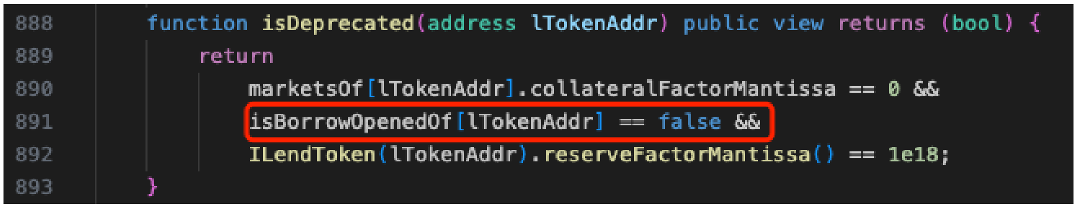
Description

When the beforeBorrow() method is executed, it will judge whether the isBorrowOpenedOf condition is satisfied. After the isBorrowOpenedOf[msg.sender] is used to judge the condition, it will continue to be judged by the isDeprecated() method. The isDeprecated() method also judges the isBorrowOpenedOf condition. If the condition of isBorrowOpenedOf[msg.sender] is satisfied, the judgment result of the isDeprecated() method here is always false, so the judgment condition can always be passed here. Therefore, the judgment condition of isBorrowOpenedOf[msg.sender] and !isDeprecated(msg.sender) are repeated.

Code location:



```
386     function beforeBorrow(
387         address borrower,
388         uint256 borrowAmount
389     ) external checkListed {
390         require(isBorrowOpenedOf[msg.sender], "Controller: borrow is closed");
391         require(
392             oracle.getUnderlyingPrice(msg.sender) > 0,
393             "Controller: price is zero"
394         );
395 
396         require(!isDeprecated(msg.sender), "Controller: market is deprecated");
397 
398         if (!marketsOf[msg.sender].accountMembership[borrower]) {
399             addToMarket(msg.sender, borrower);
400         }
401         if (borrowCaps[msg.sender] > 0) {
402             require(
403                 (ILendToken(msg.sender).totalBorrows() + borrowAmount) <= borrowCaps[msg.sender],
404                 "Controller: market borrow cap reached"
405             );
406         }
407         (, uint256 shortfall) = getHypotheticalAccountLiquidity(
408             borrower,
409             msg.sender,
410             0,
411             borrowAmount
412         );
413         require(shortfall == 0, "Controller: will be liquidated");
414         deposit(msg.sender, borrowAmount, borrower, false);
415     }
```



```
888     function isDeprecated(address lTokenAddr) public view returns (bool) {
889         return
890             marketsOf[lTokenAddr].collateralFactorMantissa == 0 &&
891             isBorrowOpenedOf[lTokenAddr] == false &&
892             ILendToken(lTokenAddr).reserveFactorMantissa() == 1e18;
893     }
```

Recommendation

It is recommended to delete one of the above two judgment conditions..

Status

confirmed.

The project party responded that it was indeed repeated judgments, and decided to modify and delete the repeated judgments.

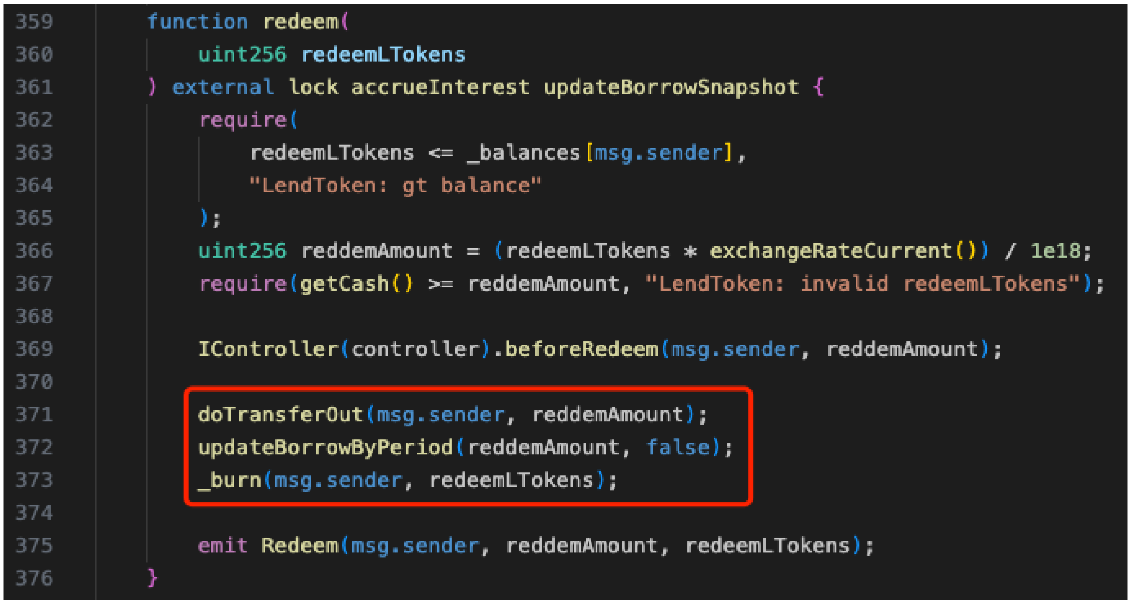
4.2.7 Funds transfer sequence is not secure

ID	Severity	Location	Status
07	Medium	TokenTemplate.sol: 359, 376	fixed

Description

When a user redeems assets, since two fund transfers are involved, it is recommended to destroy the LToken borrowed by the user first, and then transfer the corresponding amount of assets to the user to avoid reentry.

Code location:



```
359     function redeem(
360         uint256 redeemLTokens
361     ) external lock accrueInterest updateBorrowSnapshot {
362         require(
363             redeemLTokens <= _balances[msg.sender],
364             "LendToken: gt balance"
365         );
366         uint256 reddemAmount = (redeemLTokens * exchangeRateCurrent()) / 1e18;
367         require(getCash() >= reddemAmount, "LendToken: invalid redeemLTokens");
368 
369         IController(controller).beforeRedeem(msg.sender, reddemAmount);
370 
371         doTransferOut(msg.sender, reddemAmount);
372         updateBorrowByPeriod(reddemAmount, false);
373         _burn(msg.sender, redeemLTokens);
374 
375         emit Redeem(msg.sender, reddemAmount, redeemLTokens);
376     }
```

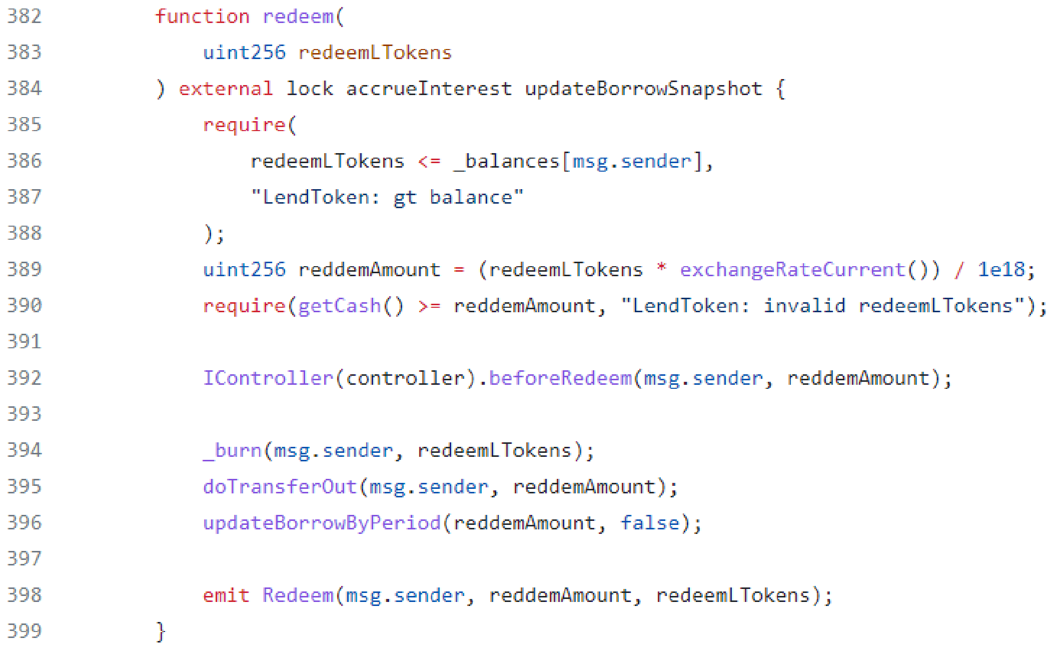
Recommendation

It is recommended to destroy the LToken borrowed by the user first, and then transfer the corresponding amount of assets to the user to avoid re-entry.

Status

fixed.

commits: f16195d1708db62fbe208cdec83b3bdb3ea45b11



```
    function redeem(
        uint256 redeemLTokens
    ) external lock accrueInterest updateBorrowSnapshot {
        require(
            redeemLTokens <= _balances[msg.sender],
            "LendToken: gt balance"
        );
        uint256 reddemAmount = (redeemLTokens * exchangeRateCurrent()) / 1e18;
        require(getCash() >= reddemAmount, "LendToken: invalid redeemLTokens");

        IController(controller).beforeRedeem(msg.sender, reddemAmount);

        _burn(msg.sender, redeemLTokens);
        doTransferOut(msg.sender, reddemAmount);
        updateBorrowByPeriod(reddemAmount, false);

        emit Redeem(msg.sender, reddemAmount, redeemLTokens);
    }
```

4.2.8 Privileged roles can update contract variables

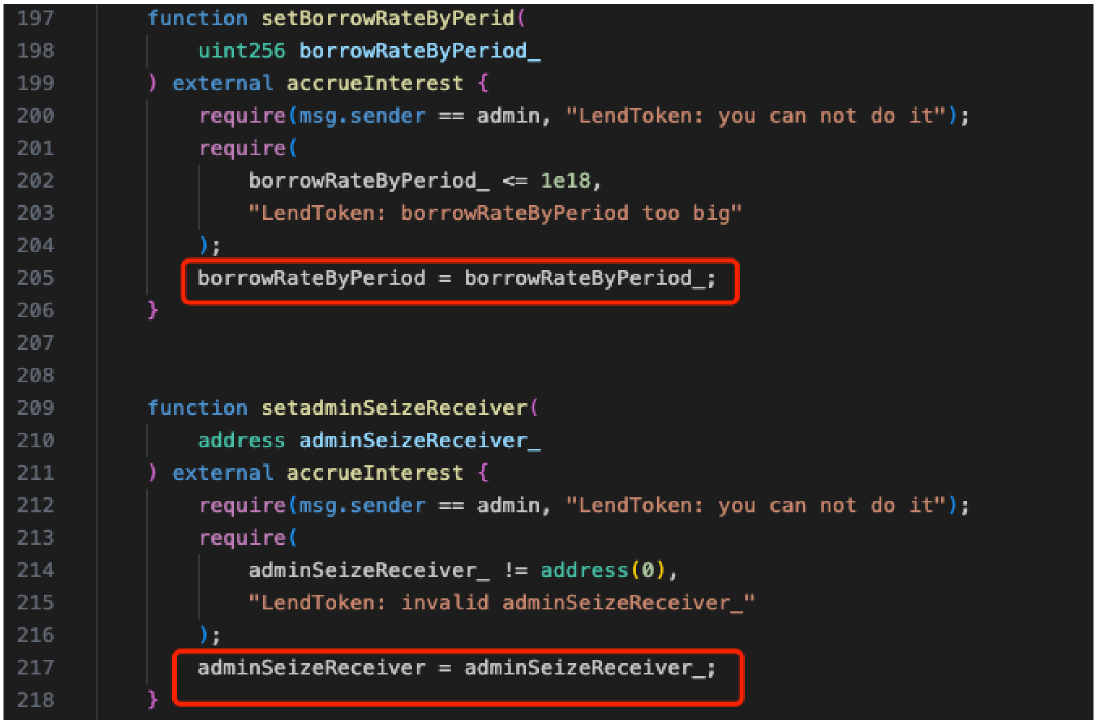
ID	Severity	Location	Status
08	Low	TokenTemplate.sol: 197, 218 OracelBridge.sol: 30, 51; 60, 80	confirmed

Description

TokenTemplate.sol

adminSeizeReceiver variable,The admin privileged role can set the maximum lending ratio of units and the receiving address of platform liquidation rewards. If the admin privileged role is maliciously controlled, it may cause the loss of project and user funds.

Code location:

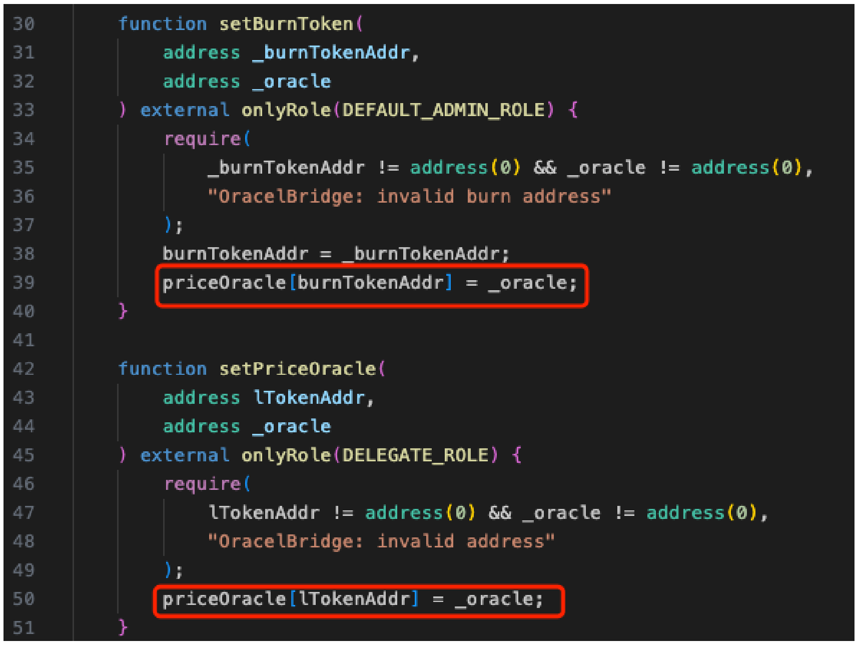


```
197     function setBorrowRateByPerid(
198         uint256 borrowRateByPeriod_
199     ) external accrueInterest {
200         require(msg.sender == admin, "LendToken: you can not do it");
201         require(
202             borrowRateByPeriod_ <= 1e18,
203             "LendToken: borrowRateByPeriod too big"
204         );
205         borrowRateByPeriod = borrowRateByPeriod_;
206     }
207
208
209     function setadminSeizeReceiver(
210         address adminSeizeReceiver_
211     ) external accrueInterest {
212         require(msg.sender == admin, "LendToken: you can not do it");
213         require(
214             adminSeizeReceiver_ != address(0),
215             "LendToken: invalid adminSeizeReceiver_"
216         );
217         adminSeizeReceiver = adminSeizeReceiver_;
218     }
```

OracelBridge.sol

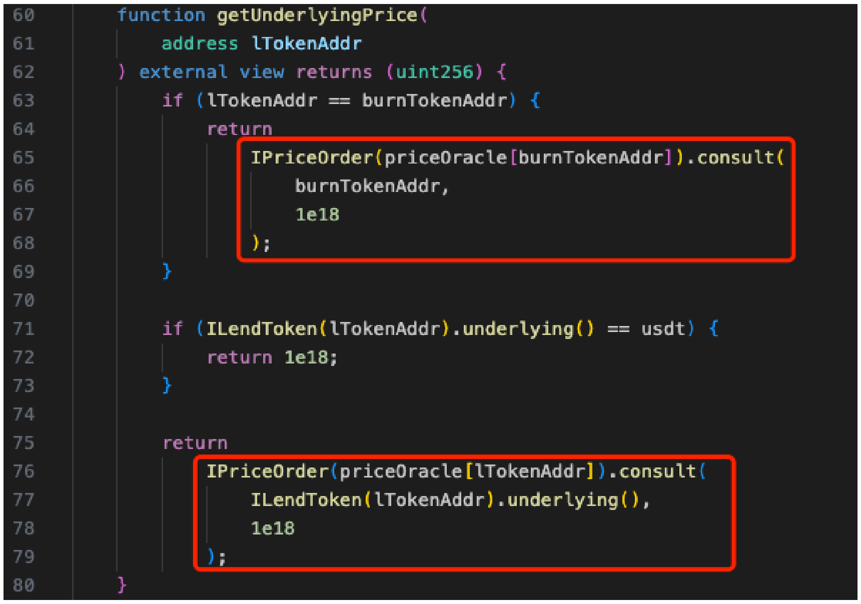
Since the DEFAULT_ADMIN_ROLE and DELEGATE_ROLE privileged roles can set priceOracle, when the privileged role is maliciously controlled, it may lead to obtaining the price in the malicious Oracle contract, resulting in the loss of project and user funds.

Code location:



```
    function setBurnToken(
        address _burnTokenAddr,
        address _oracle
    ) external onlyRole(DEFAULT_ADMIN_ROLE) {
        require(
            _burnTokenAddr != address(0) && _oracle != address(0),
            "OracleBridge: invalid burn address"
        );
        burnTokenAddr = _burnTokenAddr;
        priceOracle[burnTokenAddr] = _oracle;
    }

    function setPriceOracle(
        address lTokenAddr,
        address _oracle
    ) external onlyRole(DELEGATE_ROLE) {
        require(
            lTokenAddr != address(0) && _oracle != address(0),
            "OracleBridge: invalid address"
        );
        priceOracle[lTokenAddr] = _oracle;
    }
```



```
    function getUnderlyingPrice(
        address lTokenAddr
    ) external view returns (uint256) {
        if (lTokenAddr == burnTokenAddr) {
            return
                IPriceOrder(priceOracle[burnTokenAddr]).consult(
                    burnTokenAddr,
                    1e18
                );
        }

        if (ILendToken(lTokenAddr).underlying() == usdt) {
            return 1e18;
        }

        return
            IPriceOrder(priceOracle[lTokenAddr]).consult(
                ILendToken(lTokenAddr).underlying(),
                1e18
            );
    }
```

Recommendation

It is recommended that privileged roles be managed using multi-signatures and timelocks.

Status

Confirmed.

5 Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Disclaimer

This report is issued in response to facts that occurred or existed prior to the issuance of this report, and liability is assumed only on that basis.
Shield Security cannot determine the security status of this program and assumes no responsibility for facts occurring or existing after the date of this report. The security audit analysis and other content in this report is based on documents and materials provided to Shield Security by the information provider through the date of the insurance report. in Shield Security's opinion. The information provided is not missing, falsified, deleted or concealed. If the information provided is missing, altered, deleted, concealed or not in accordance with the actual circumstances, Shield Security shall not be liable for any loss or adverse effect resulting therefrom. shield Security will only carry out the agreed security audit of the security status of the project and issue this report. shield Security is not responsible for the background and other circumstances of the project. Shield Security is not responsible for the background and other circumstances of the project.