Fortube Bank 2.0

Smart Contract Security Audit Report

2020-09-04

1. Executive Summary....3
2. Audit Methodology....4
3 Project Background(Context)....5
3.1 Project Introduction....5
3.2 Project Structure....6
3.3 Contract Structure....7
4. Code Overview....8
4.1 Main file hash....8
4.2 Main function visibility analysis....13
5 Audit Result....29
5.1 Critical Vulnerabilities....29
5.1.1 Authentication check bypassing....29
5.2 High Risk Vulnerabilities....30
5.3 Medium Risk Vulnerabilities....30
5.3.1 Not set gas limit to external call....30
5.3.2 Missing check....31
5.3.3 Precision issue....35
5.3.4 System unavailability risk....36
5.4 Low Risk Vulnerabilities....39
5.4.1 Missing check....39
5.5 Audit conclusion....42

5.5.1 Critical Vulnerabilities 42
5.5.2 Medium Risk vulnerabilities 43
5.5.3 Low Risk vulnerabilities 43
5.5.4 Conclusion 43
6 Statement 44

1. Executive Summary

On Aug. 25, 2020, the SlowMist security team received the Fortube team's security audit application for Fortube Bank 2.0 system, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

SlowMist Smart Contract DeFi project test method:

Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code module through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

SlowMist Smart Contract DeFi project risk level:

Critical vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High-risk vulnerabilities	High-risk vulnerabilities will affect the normal operation of DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium-risk vulnerablities	Medium vulnerability will affect the operation of DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low-risk vulnerabilities	Low-risk vulnerabilities may affect the operation of DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weaknesses	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Enhancement Suggestions	There are better practices for coding or architecture.

2. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and in-house automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look

for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Reentrancy attack and other Race Conditions
- Replay attack
- Reordering attack
- Short address attack
- Denial of service attack
- Transaction Ordering Dependence attack
- Conditional Completion attack
- Authority Control attack
- Integer Overflow and Underflow attack
- TimeStamp Dependence attack
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Explicit visibility of functions state variables
- Logic Flaws
- Uninitialized Storage Pointers
- Floating Points and Numerical Precision
- tx.origin Authentication
- "False top-up" Vulnerability
- Scoping and Declarations

3 Project Background

3.1 Project Introduction

ForTube is the world's top DeFi lending platform launched by The Force Protocol. It is committed to providing decentralized lending services for cryptoasset enthusiasts around the

world, supporting most of the world's popular assets. ForTube is based on smart contracts and automated algorithm technology. Users can deposit tokens to earn interest, pledge and borrow tokens to pay interests. ForTube's interest rate is determined by market supply and demand. Assets are controlled by users. ForTube allows users to deposit and withdraw anytime, borrow and repay anytime globally.

We audited Fortube Bank 2.0 smart contract code, the following is the relevant file information:

Audit version file information

Initial audit files:

Fortube2.zip(SHA256):

14c501812d7a53ef5b7ded2e9540d7bc07967a1ef6c9e9cadfb266f1d99b062e

Final audit files:

Fortube2.zip(SHA256):

67b9d5f8c88cb3cf7ccfd115c487512fd72e73a7ee83cd16106193379aa9e1a8

3.2 Project Structure

contracts

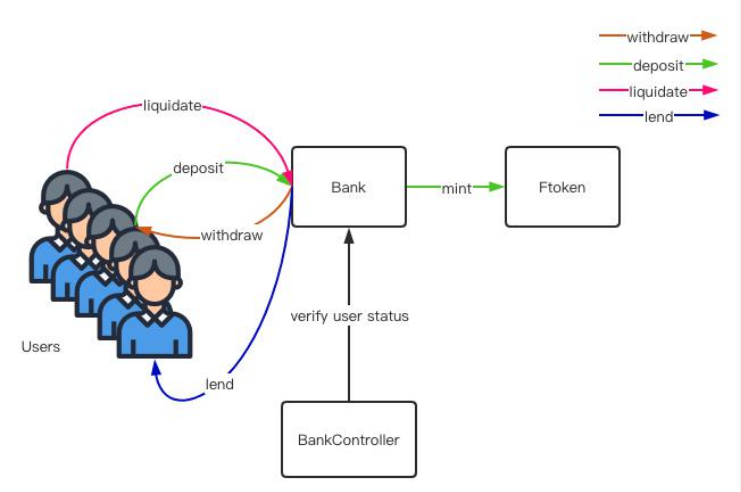
```
├── Bank.sol
├── BankController.sol
├── Broker.sol
├── Exponential.sol
├── FToken.sol
├── InterestRateModel.sol
├── MSign.sol
├── Migrations.sol
├── MonitorEventMock.sol
├── PriceOracles.sol
├── RewardPool.sol
├── RewardType.sol
```

```
├── fTokens
│   ├── fBUSD.sol
│   ├── fDAI.sol
│   ├── fETH.sol
│   ├── fFOR.sol
│   ├── fHBTC.sol
│   ├── fHUSD.sol
│   ├── fUSDC.sol
│   └── fUSDT.sol
├── interface
│   ├── IBank.sol
│   ├── IBankController.sol
│   ├── IERC20.sol
│   ├── IFToken.sol
│   ├── IInterestRateModel.sol
│   ├── IOracle.sol
│   └── IRewardPool.sol
├── library
│   ├── Address.sol
│   ├── EthAddressLib.sol
│   ├── SafeERC20.sol
│   ├── SafeMath.sol
│   └── WadRayMath.sol
└── mock
    ├── MockETHChainLink.sol
    ├── MockOracle.sol
    └── MockUSDT.sol
```

3.3 Contract Structure

Fortube Bank 2.0 is a lending protocol that allows user to deposit assets as collaterals to pools and lend other assets from other pools. The Bank contract is the entrance of the whole system and implements the main function such as deposit, withdraw, liquidate etc. The BankController contract is used to verify user status and store all funds deposited from users, included user fund and borrow interest. Ftoken contract is used to mint Ftokens when user deposit, and will accumulate value from borrow interest.

The main structure is as following:



4. Code Overview

4.1 Main Contract address

Bank	
Proxy contract address	0xdE7B3b2Fe0E7b4925107615A5b199a4EB40D9ca9
Logic contract address	0xb6d6209b5e2c3f827beb1b67bd908ec06c9028ba

BankController	
Proxy contract address	0x936E6490eD786FD0e0f0C1b1e4E1540b9D41F9eF
Logic contract address	0x660364d12261ca3bd28079fdb61583651a44124b

FBNB	
Proxy contract address	0x92563b3b8c92B22e37aC956a2B19c40988D25933
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

FBUSD	
Proxy contract address	0x556be90ea81e8abceEc2737cf6AE0a6cfEe58b40
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

FDAI	
Proxy contract address	0xfF5cDA31926CA2Ed79533D6B95Fc6ADbDE0f1015
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

FETH	
Proxy contract address	0x5993233d88B4424D9c12e468A39736D5948c2835
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

FFOR	
Proxy contract address	0x84ff569ee2E8b9A2C22E79af431fD248fb41D87b
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

FHBTC	
Proxy contract address	0x3CE92b88DEAec1037335E614Eb6409C95edcAC76
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

FHT	
Proxy contract address	0x39527B067B04D43c627FB741848ef2c3f8ead3FE
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

FHUSD	
Proxy contract address	0x52d61a0AA88170b6EbDEA25Be1561E5665e6481B
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

FOKB	
Proxy contract address	0x4316AAa55ab3BD3a7ee3fbC83580521801210225
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

FQC	
Proxy contract address	0x161190d29cC015EaEFD6c4ad0AA7519B6b75b9c0
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

FUSDC	
Proxy contract address	0xfDD543Ed2701dEB2a172Df4876E60918E28Ba217
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

FUSDT	
Proxy contract address	0x51da0A7340874674C488b67200D007E422667650
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

FWBTC	
Proxy contract address	0x93B9B852FcD2f964Faa7E50EC1374e016260718c
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

FYUSD	
Proxy contract address	0x66142B3c234C054bA91374732C10cEA0f72390fE
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

FYWETH	
Proxy contract address	0xd136b32330E539aa9411c4e8968eB26b35c5917B
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

FYFII	
Proxy contract address	0x4Ffc92ddD9439c93fc79dD5560e06026A445037D
Logic contract address	0x322401fff76dda9cea5595aaa90d062798907bde

InterestRateModel	
Proxy contract address	/
Logic contract address	0x341661a71bbCfEA48CdcF0A104aF3e708aABeBf9

Msign	
Proxy contract address	/
Logic contract address	0xf8345037Da48e90A68A9590C4bBAad6fbbd62661

PriceOracles	
Proxy contract address	/
Logic contract address	0xC05E28cb46605778A247A6e25aB016897527979d

RewardPool	
Proxy contract address	0x775f9099e24CD67f75232245719f363c0243783F
Logic contract address	0x7fd534f692050dd0ca6e2113cb953eb630d6dcfd

4.2 Main function visibility analysis

Contract	Function Name	Visibility
BankController	setBankEntryAddress	External
	setTheForceToken	External
	setRewardFactorByType	External
	marketsContains	Public
	initialize	Public
	proposeNewAdmin	External
	claimAdministration	External
	getFTokeAddress	Public
	getAssetsIn	External
	checkAccountsIn	External
	userEnterMarket	Internal
	transferCheck	External
	withdrawCheck	Public
	transferIn	Public
	transferToUser	Public
	transferToUserInternal	Internal

	calcRewardAmount	Public
	calcRewardAmountByFactor	Public
	setRewardPool	External
	setTransferEthGasCost	External
	rewardForByType	External
	getCashPrior	Public
	getCashAfter	Public
	mintCheck	Public
	borrowCheck	Public
	repayCheck	Public
	getTotalDepositAndBorrow	Public
	getAccountLiquidity	Public
	getAccountLiquidityExcludeDeposit	Public
	fetchAssetPrice	Public
	setOracle	Public
	_supportMarket	External
	addTokenToMarket	Internal

	_setCollateralAbility	External
	setCloseFactor	External
	getAllMarkets	Public
	seizeCheck	External
	getUserLiquidity	Public
	getUserLiquidityExcludeToken	Public
	tokenDecimals	Public
	calcMaxWithdrawAmount	Public
	calcMaxBorrowAmount	Public
	calcMaxBorrowAmountWithRatio	Public
	calcMaxCashOutAmount	Public
	isFTokenValid	External
	liquidateBorrowCheck	External
	calcExchangeUnit	Public
	liquidateTokens	External
	_setLiquidationIncentive	Public

	reduceReserves	Public
	addReserves	Public
EthAddressLib	ethAddress	Internal
Exponential	getExp	Public
	getDiv	Public
	addExp	Public
	subExp	Public
	mulExp	Public
	divExp	Public
	mulExp3	Public
	mulScalar	Public
	mulScalarTruncate	Public
	mulScalarTruncateAddUInt	Public
	divScalarByExpTruncate	Public
	divScalarByExp	Public
	divScalar	Public
	truncate	Public
SafeMath	add	Internal

	sub	Internal
	sub	Internal
	mul	Internal
	div	Internal
	div	Internal
	mod	Internal
	mod	Internal
	min	Internal
	abs	Internal
WadRayMath	ray	Internal
	wad	Internal
	halfRay	Internal
	halfWad	Internal
	wadMul	Internal
	wadDiv	Internal
	rayMul	Internal
	rayDiv	Internal
	rayToWad	Internal
	wadToRay	Internal
	rayPow	Internal

IFToken	Interface	IERC20
	mint	External
	borrow	External
	withdraw	External
	underlying	External
	accrueInterest	External
	getAccountState	External
	MonitorEventCallback	External
	exchangeRateCurrent	External
	repay	External
	borrowBalanceStored	External
	exchangeRateStored	External
	liquidateBorrow	External
	borrowBalanceCurrent	External
	balanceOfUnderlying	External
	_reduceReserves	External
	_addReservesFresh	External
	cancellingOut	External
	APR	External

	APY	External
	calcBalanceOfUnderlying	External
	borrowSafeRatio	External
	tokenCash	External
	getBorrowRate	External
	addTotalCash	External
	subTotalCash	External
	totalCash	External
IERC20	totalSupply	External
	balanceOf	External
	transfer	External
	allowance	External
	approve	External
	transferFrom	External
	decimals	External
IOracle	get	External
SafeERC20	safeTransfer	Internal
	safeTransferFrom	Internal
	safeApprove	Internal

	safeIncreaseAllowance	Internal
	safeDecreaseAllowance	Internal
	_callOptionalReturn	Private
Address	isContract	Internal
	sendValue	Internal
	functionCall	Internal
	functionCall	Internal
	functionCallWithValue	Internal
	functionCallWithValue	Internal
	_functionCallWithValue	Private
IBank	MonitorEventCallback	External
	deposit	External
	borrow	External
	withdraw	External
	withdrawUnderlying	External
	repay	External
	liquidateBorrow	External
	tokenIn	External

	tokenOut	External
	cancellingOut	External
IRewardPool	theForceToken	External
	bankController	External
	admin	External
	deposit	External
	withdraw	External
	withdraw	External
	setTheForceToken	External
	setBankController	External
	reward	External
FToken	initialize	Public
	_setController	Public
	tokenCash	Public
	transfer	External
	transferFrom	External
	transferTokens	Internal
	approve	External
	allowance	External

	mint	Public
	mintInternal	Internal
	borrow	Public
	borrowInternal	Internal
	exchangeRateStored	Public
	calcExchangeRate	Public
	exchangeRateAfter	Public
	balanceOfUnderlying	Public
	calcBalanceOfUnderlying	Public
	exchangeRateCurrent	Public
	getAccountState	External
	withdraw	External
	withdrawInternal	Internal
	strikeWithdrawInternal	Internal
	accrueInterest	Public
	peekInterest	Public
	borrowBalanceCurrent	Public

	borrowBalanceStoredInternal	Internal
	_setReserveFactorFresh	Public
	_setInterestRateModel	Public
	getBlockNumber	Internal
	repay	Public
	repayInternal	Internal
	borrowBalanceStored	Public
	liquidateBorrow	Public
	seize	External
	cancellingOut	Public
	balanceOf	Public
	_setBorrowSafeRatio	Public
	seizeInternal	Internal
	_reduceReserves	External
	_addReservesFresh	External
	addTotalCash	Public

	subTotalCash	Public
	APR	Public
	APY	External
	getBorrowRate	Public
	getSupplyRate	Public
IInterestRateModel	utilizationRate	External
	getBorrowRate	External
	getSupplyRate	External
	APR	External
	APY	External
IBankController	getCashPrior	External
	getCashAfter	External
	getFTokeAddress	External
	transferToUser	External
	transferIn	External
	borrowCheck	External
	repayCheck	External
	rewardForByType	External
	liquidateBorrowCheck	External

	liquidateTokens	External
	withdrawCheck	External
	transferCheck	External
	marketsContains	External
	seizeCheck	External
	mintCheck	External
	addReserves	External
	reduceReserves	External
	calcMaxBorrowAmount	External
	calcMaxWithdrawAmount	External
	calcMaxCashOutAmount	External
	calcMaxBorrowAmountWithRatio	External
	transferEthGasCost	External
	isFTokenValid	External
Bank	MonitorEventCallback	External
	initialize	Public
	setController	Public

	setPaused	Public
	setUnpaused	Public
	proposeNewAdmin	External
	claimAdministration	External
	deposit	Public
	_deposit	External
	borrow	Public
	withdraw	Public
	withdrawUnderlying	Public
	repay	Public
	_repay	Public
	liquidateBorrow	Public
	tokenIn	Public
	tokenOut	External
	cancellingOut	Public

Broker	initialize	Public
	version	Public
	subscribe	Public
	unsubscribe	Public
	publish	External
	subscribers	External
InterestRateModel	Constructor	Public
	utilizationRate	Public
	getBorrowRate	Public
	getSupplyRate	Public
	APR	External
	APY	External
Msign	Constructor	Public
	gethash	Public
	activate	Public
	execute	Public
	sign	Public
	enable	Public
	disable	Public
	mulsignweight	Public

	threshold	Public
	signers	Public
	signable	Public
Migrations	setCompleted	Public
AggregatorInterface	latestAnswer	External
IUniversalOracle	get	External
PriceOracles	get	External
	Constructor	Public
	setEthToUsdPrice	External
	setOracle	External
	proposeNewAdmin	External
	claimAdministration	External
	setTokenChainlinkMap	External
	getChainLinkPrice	Internal
RewardPool	initialize	Public
	deposit	External
	withdraw	External
	withdraw	External
	setTheForceToken	External

	setBankController	External
	reward	External
fBUSD	initialize	Public
fDAI	initialize	Public
fETH	initialize	Public
fFOR	initialize	Public
fHBTC	initialize	Public
fHUSD	initialize	Public
fUSDC	initialize	Public
fUSDT	initialize	Public

5 Audit Result

5.1 Critical Vulnerabilities

Critical severity issues can have a major impact on the security of smart contracts, and it is highly recommended to fix critical severity vulnerability.

5.1.1 Authentication check bypassing

(1) The transferToUser function on BankController is public, which causes the msg.sender in the function to be a contract to forge the underlying to return a list of valid tokens, leading to bypassing

the permission check and transferring any tokens.

Location: BankController.sol Line: 218

```
function transferToUser(
        address underlying,
        address payable account,
        uint256 amount
    ) public {
        require(
            markets[IFToken(msg.sender).underlying()].isValid,
            "TransferToUser not allowed"
        );
        transferToUserInternal(underlying, account, amount);
    }
```

Fixed status: Fixed.

5.2 High Risk Vulnerabilities

High severity issues can affect the normal operation of smart contracts, and it is highly recommended to fix high severity vulnerability.

The audit has shown no high severity vulnerability.

5.3 Medium Risk Vulnerabilities

Medium severity issues can affect the operation of a smart contract, and it is recommended to fix medium severity vulnerability.

5.3.1 Not set gas limit to external call

(1) The transferIn function of the BankController contract did not limit the gas to 2300 when returning user assets, resulting in users being able to use gas in other places, and this part of gas can

also be rewarded in tokens.

Location: BankController.sol Line: 211

```
    function transferIn(
        address account,
        address underlying,
        uint256 amount
    ) public payable {
        if (underlying != EthAddressLib.ethAddress()) {
            require(msg.value == 0, "ERC20 do not accecpt ETH.");
            uint256 balanceBefore = IERC20(underlying).balanceOf(address(this));
            IERC20(underlying).safeTransferFrom(account, address(this), amount);
            uint256 balanceAfter = IERC20(underlying).balanceOf(address(this));
            require(
                balanceAfter - balanceBefore == amount,
                "TransferIn amount not valid"
            );
            // erc 20 => transferFrom
        } else {
            // 接收 eth 转账，已经通过 payable 转入
            require(msg.value >= amount, "Eth value is not enough");
            if (msg.value > amount) {
                //send back excess ETH
                uint256 excessAmount = msg.value.sub(amount);
                //solium-disable-next-line
                (bool result, ) = account.call{value: excessAmount}("");
                require(result, "Transfer of ETH failed");
            }
        }
    }
```

Fix status: fixed.

5.3.2 Missing check

(1) The calculation of the borrowRate limit by the peekInterest function and accrueInterest function in the Ftoken contract is checked before updating the new loan amount, and the

corresponding check should be moved to the borrow amount update and then checked.

Location: Ftoken.sol Line: 769, Line:704

```
function peekInterest()
        public
        view
        returns (
            uint256 _accrualBlockNumber,
            uint256 _borrowIndex,
            uint256 _totalBorrows,
            uint256 _totalReserves
        )
    {
        _accrualBlockNumber = getBlockNumber();
        uint256 accrualBlockNumberPrior = accrualBlockNumber;

        // 太短 零利息
        if (accrualBlockNumberPrior == _accrualBlockNumber) {
            return (
                accrualBlockNumber,
                borrowIndex,
                totalBorrows,
                totalReserves
            );
        }

        uint256 cashPrior = controller.getCashPrior(underlying);
        uint256 borrowsPrior = totalBorrows;
        uint256 reservesPrior = totalReserves;
        uint256 borrowIndexPrior = borrowIndex;

        // // 计算借贷利率
        uint256 borrowRate = interestRateModel.getBorrowRate(
            cashPrior,
            borrowsPrior,
            reservesPrior
        );
        // // 不能超过最大利率
        require(borrowRate <= borrowRateMax, "borrow rate is absurdly high");
```

```
        // // 计算块差
        uint256 blockDelta = _accrualBlockNumber.sub(accrualBlockNumberPrior);

        /*          *  simpleInterestFactor = borrowRate * blockDelta          *  interestAccumulated =
simpleInterestFactor * totalBorrows          *  totalBorrowsNew = interestAccumulated + totalBorrows          *
totalReservesNew = interestAccumulated * reserveFactor + totalReserves          *  borrowIndexNew =
simpleInterestFactor * borrowIndex + borrowIndex          */

        uint256 simpleInterestFactor;
        uint256 interestAccumulated;
        uint256 totalBorrowsNew;
        uint256 totalReservesNew;
        uint256 borrowIndexNew;

        simpleInterestFactor = mulScalar(borrowRate, blockDelta);

        interestAccumulated = divExp(
            mulExp(simpleInterestFactor, borrowsPrior),
            expScale
        );

        totalBorrowsNew = addExp(interestAccumulated, borrowsPrior);

        totalReservesNew = addExp(
            divExp(mulExp(reserveFactor, interestAccumulated), expScale),
            reservesPrior
        );

        borrowIndexNew = addExp(
            divExp(mulExp(simpleInterestFactor, borrowIndexPrior), expScale),
            borrowIndexPrior
        );

        _borrowIndex = borrowIndexNew;
        _totalBorrows = totalBorrowsNew;
        _totalReserves = totalReservesNew;

        //SlowMist// A borrow rate check should be added here
    }
```

```
function accrueInterest() public onlyRestricted {
        uint256 currentBlockNumber = getBlockNumber();
        uint256 accrualBlockNumberPrior = accrualBlockNumber;

        // 太短 零利息
        if (accrualBlockNumberPrior == currentBlockNumber) {
            return;
        }

        uint256 cashPrior = controller.getCashPrior(underlying);
        uint256 borrowsPrior = totalBorrows;
        uint256 reservesPrior = totalReserves;
        uint256 borrowIndexPrior = borrowIndex;

        // // 计算借贷利率
        uint256 borrowRate = interestRateModel.getBorrowRate(
            cashPrior,
            borrowsPrior,
            reservesPrior
        );
        // // 不能超过最大利率
        require(borrowRate <= borrowRateMax, "borrow rate is absurdly high");

        // // 计算块差
        uint256 blockDelta = currentBlockNumber.sub(accrualBlockNumberPrior);

        /*          *  simpleInterestFactor = borrowRate * blockDelta          *  interestAccumulated =
simpleInterestFactor * totalBorrows          *  totalBorrowsNew = interestAccumulated + totalBorrows          *
totalReservesNew = interestAccumulated * reserveFactor + totalReserves          *  borrowIndexNew =
simpleInterestFactor * borrowIndex + borrowIndex          */

        uint256 simpleInterestFactor;
        uint256 interestAccumulated;
        uint256 totalBorrowsNew;
        uint256 totalReservesNew;
        uint256 borrowIndexNew;

        simpleInterestFactor = mulScalar(borrowRate, blockDelta);
```

```
        interestAccumulated = divExp(
            mulExp(simpleInterestFactor, borrowsPrior),
            expScale
        );

        totalBorrowsNew = addExp(interestAccumulated, borrowsPrior);

        totalReservesNew = addExp(
            divExp(mulExp(reserveFactor, interestAccumulated), expScale),
            reservesPrior
        );

        borrowIndexNew = addExp(
            divExp(mulExp(simpleInterestFactor, borrowIndexPrior), expScale),
            borrowIndexPrior
        );

        accrualBlockNumber = currentBlockNumber;
        borrowIndex = borrowIndexNew;
        totalBorrows = totalBorrowsNew;
        totalReserves = totalReservesNew;

        //SlowMist// A borrow rate check should be added here
    }
```

Fix status: Fixed.

5.3.3 Precision issue

The withdrawCheck on BankController contract does not perform precision processing on Ftoken, which causes calculation problems when precision of ftoken and underlying are inconsistent.

```
function withdrawCheck(
        address fToken,
        address withdrawer,
        uint256 withdrawTokens
    ) public view returns (uint256) {
        require(
```

```
            markets[IFToken(fToken).underlying()].isValid,
            "Market not valid"
        );

        (uint256 sumCollaterals, uint256 sumBorrows) = getUserLiquidity(
            withdrawer,
            IFToken(fToken),
            withdrawTokens,
            0
        );
        require(SafeMath.sub(sumCollaterals, sumBorrows) >= 0, "Cannot withdraw tokens");
    }
```

Fixed status: Fixed.

5.3.4 System unavailability risk

The accrueInterest function has a problem in processing borrawRate. Because the restriction on borrowRate is to use require, the following functions cannot be called normally when the borrowing rate is really higher than the maximum borrowing rate.

- Ftoken contract _setReserveFactorFresh function
- Ftoken contract _setInterestRateModel function
- Ftoken contract repay function
- Ftoken contract cancellingOut function
- Ftoken contract _reduceReserves function
- Ftoken contract _addReservesFresh function
- Ftoken contract exchangeRateCurrent function

- Ftoken contract borrowBalanceCurrent function
- Ftoken contract liquidateBorrow function

e.g Ftoken contract exchangeRateCurrent function

Location: Ftoken.sol Line: 515

```
    function exchangeRateCurrent() public nonReentrant returns (uint256) {
        accrueInterest();
        return exchangeRateStored();
    }
```

```
function accrueInterest() public onlyRestricted {
        uint256 currentBlockNumber = getBlockNumber();
        uint256 accrualBlockNumberPrior = accrualBlockNumber;

        // 太短 零利息
        if (accrualBlockNumberPrior == currentBlockNumber) {
            return;
        }

        uint256 cashPrior = controller.getCashPrior(underlying);
        uint256 borrowsPrior = totalBorrows;
        uint256 reservesPrior = totalReserves;
        uint256 borrowIndexPrior = borrowIndex;

        // // 计算借贷利率
        uint256 borrowRate = interestRateModel.getBorrowRate(
            cashPrior,
            borrowsPrior,
            reservesPrior
        );
        // // 不能超过最大利率
        require(borrowRate <= borrowRateMax, "borrow rate is absurdly high");

        // // 计算块差
        uint256 blockDelta = currentBlockNumber.sub(accrualBlockNumberPrior);
```

```
        /*          *  simpleInterestFactor = borrowRate * blockDelta          *  interestAccumulated =
simpleInterestFactor * totalBorrows          *  totalBorrowsNew = interestAccumulated + totalBorrows          *
totalReservesNew = interestAccumulated * reserveFactor + totalReserves          *  borrowIndexNew =
simpleInterestFactor * borrowIndex + borrowIndex          */

        uint256 simpleInterestFactor;
        uint256 interestAccumulated;
        uint256 totalBorrowsNew;
        uint256 totalReservesNew;
        uint256 borrowIndexNew;

        simpleInterestFactor = mulScalar(borrowRate, blockDelta);

        interestAccumulated = divExp(
            mulExp(simpleInterestFactor, borrowsPrior),
            expScale
        );

        totalBorrowsNew = addExp(interestAccumulated, borrowsPrior);

        totalReservesNew = addExp(
            divExp(mulExp(reserveFactor, interestAccumulated), expScale),
            reservesPrior
        );

        borrowIndexNew = addExp(
            divExp(mulExp(simpleInterestFactor, borrowIndexPrior), expScale),
            borrowIndexPrior
        );

        accrualBlockNumber = currentBlockNumber;
        borrowIndex = borrowIndexNew;
        totalBorrows = totalBorrowsNew;
        totalReserves = totalReservesNew;
    }
```

Fix status: After communicating with the project party, it is confirmed that the risk here is known, but according to the current interest rate model design, this risk cannot be triggered.

5.4 Low Risk Vulnerabilities

Low severity issues can affect smart contracts operation in future versions of code. We recommend the project party to evaluate and consider whether these problems need to be fixed.

5.4.1 Missing check

(1) Bank contract does not check if Ftoken is in the market in the deposit function.

Location: Bank.sol Line: 112

```
    function deposit(address token, uint256 amount)
        public
        payable
        whenUnpaused
        rewardFor(msg.sender, RewardType.Deposit)
    {
        //SlowMist// A Ftoken check should be added here

        IFToken fToken = IFToken(controller.getFTokeAddress(token));
        bytes memory flog = fToken.mint(msg.sender, amount);
        controller.transferIn{value: msg.value}(msg.sender, token, amount);
        emit MonitorEvent("Deposit", flog);
    }
```

Fix status: Fixed.

(2) Bank contract does not check if Ftoken is in the market in the cashIn function.

Location: Bank.sok Line: 214

```
function cashIn(address token, uint256 amountIn)
        public
        payable
        whenUnpaused
        rewardFor(msg.sender, RewardType.CashIn)
    {
        IFToken fToken = IFToken(controller.getFTokeAddress(token));
```

```
        //SlowMist// A Ftoken check should be added here

        //先进行冲账操作
        (bool strikeOk, bytes memory strikeLog) = fToken.cancellingOut(
            msg.sender
        );
        (
            bool repayOk,
            bytes memory repayLog,
            bool depositOk,
            bytes memory depositLog
        ) = fToken.cashIn(msg.sender, amountIn);
        controller.transferIn{value: msg.value}(msg.sender, token, amountIn);

        if (strikeOk) {
            emit MonitorEvent("CancellingOut", strikeLog);
        }

        if (repayOk) {
            emit MonitorEvent("Repay", repayLog);
        }
        if (depositOk) {
            emit MonitorEvent("Deposit", depositLog);
        }

        emit MonitorEvent("CashIn", abi.encode(token, amountIn));
    }
```

Fix status: Fixed.

(3) In Ftoken, accrueinterest() is not used to correct the interest during transferTokens, resulting in incorrect calculations during transfer.

Location: Ftoken.sol Line:206

```
function transferTokens(
        address spender,
        address src,
        address dst,
        uint256 tokens
    ) internal returns (bool) {
```

```
        controller.transferCheck(address(this), src, tokens);
        //SlowMist// should add accrueinterest() here to correct the interest
        require(src != dst, "Cannot transfer to self");

        uint256 startingAllowance = 0;
        if (spender == src) {
            startingAllowance = uint256(-1);
        } else {
            startingAllowance = transferAllowances[src][spender];
        }

        uint256 allowanceNew;
        uint256 srcTokensNew;
        uint256 dstTokensNew;

        allowanceNew = startingAllowance.sub(tokens);
        srcTokensNew = accountTokens[src].sub(tokens);
        dstTokensNew = accountTokens[dst].add(tokens);

        accountTokens[src] = srcTokensNew;
        accountTokens[dst] = dstTokensNew;

        if (startingAllowance != uint256(-1)) {
            transferAllowances[src][spender] = allowanceNew;
        }

        emit Transfer(src, dst, tokens);
        return true;
    }
```

Fix status: After confirming with the project party, this error is within the acceptable range.

(4) The Bank contract did not check whether liquidateBorrow and liquidateCollateral are address(0) or whether the corresponding address already exists during liquidation.

Location: Bank.sol Line:182

```
function liquidateBorrow(
        address borrower,
        address underlyingBorrow,
```

```
        address underlyingCollateral,
        uint256 repayAmount
    ) public payable whenUnpaused {
        require(msg.sender != borrower, "Liquidator cannot be borrower");
        require(repayAmount > 0, "Liquidate amount not valid");

        IFToken fTokenBorrow = IFToken(
            controller.getFTokeAddress(underlyingBorrow)
        );
        IFToken fTokenCollateral = IFToken(
            controller.getFTokeAddress(underlyingCollateral)
        );
        //SlowMist// A check should be added here to check if fTokenCollateral orTokenBorrow is
valid
        bytes memory flog = fTokenBorrow.liquidateBorrow(
            msg.sender,
            borrower,
            repayAmount,
            address(fTokenCollateral)
        );
        controller.transferIn{value: msg.value}(
            msg.sender,
            underlyingBorrow,
            repayAmount
        );

        emit MonitorEvent("LiquidateBorrow", flog);
    }
```

Fix status: fixed.

5.5 Audit conclusion

5.5.1 Critical Vulnerabilities

- Authentication check bypassing

5.5.2 Medium Risk vulnerabilities

- Not set gas limit to external call
- Missing check
- Precision issue
- System unavailability

5.5.3 Low Risk vulnerabilities

- Missing check

5.5.4 Conclusion

Audit Result : Passed

Audit Number : 0X002009040001

Audit Date : Sep. 04, 2020

Audit Team : SlowMist Security Team

Summary conclusion: The are nine security issues found during the audit. One critical risk issue, four medium risk issues, and four low risk issues. After communication and feedback with the Fortube team, confirms that the risks found in the audit process are within the tolerable range.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility base on these. For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of the smart contract, and is not responsible for it. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of this report (referred to as "the provided information"). If the provided information is missing, tampered, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom.

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account