Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.10.15, the SlowMist security team received the Filda team's security audit application for Filda 2.0, 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.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules 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.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

Level	Description
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using 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 Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Short Address Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Uninitialized Storage Pointers Vulnerability
- Arithmetic Accuracy Deviation Vulnerability
- tx.origin Authentication Vulnerability

- "False top-up" Vulnerability
- Variable Coverage Vulnerability
- Gas Optimization Audit
- Malicious Event Log Audit
- Redundant Fallback Function Audit
- Unsafe External Call Audit
- Explicit Visibility of Functions State Variables Aduit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Audit of only the iterations based on version 1.0

Audit Version:

https://github.com/fildaio/compound-protocol/tree/filda_2.0

commit: 57616f29bf95f582f05450ccf0199733ea81168b

Fixed Version:

https://github.com/fildaio/compound-protocol/tree/filda_2.0

commit: 17674bbb3c4a339c924cef93552a6a6602739f36

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Native token receiving issue	Others	Suggestion	Confirmed
N2	Missing event record	Others	Suggestion	Fixed
N3	Code redundancy issue	Others	Suggestion	Confirmed
N4	Flashloan issue	Design Logic Audit	Critical	Fixed
N5	Potential calculation flaws in flashloan fees	Design Logic Audit	Suggestion	Confirmed

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

ChainlinkAdaptor			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getUnderlyingPrice	External	-	-
getUnderlyingPriceFromFallback	Public	-	-

ChainlinkAdaptor			
getUnderlyingPriceFromChainlink	Internal	-	-
preCheckPrice	External	-	-
getSourcePrice	Public	-	-
getPrice	Public	-	-
setAssetSources	External	Can Modify State	onlyGovernance
setFallbackPriceOracle	External	Can Modify State	onlyGovernance
_setAssetsSources	Internal	Can Modify State	-

DefaultHecoInterestModel			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
utilizationRate	Public	-	-
getBorrowRate	Public	-	-
getSupplyRate	Public	-	-

QsBorrowCapCErc20Delegate			
Function Name	Visibility	Mutability	Modifiers
_setBorrowCap	Public	Can Modify State	-
borrowInternal	Internal	Can Modify State	nonReentrant

HecoJumpInterestModel

HecoJumpInterestModel			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
utilizationRate	Public	-	-
getBorrowRate	Public	-	-
getSupplyRate	Public	-	-

QsConfig			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_setMarketBorrowCaps	External	Can Modify State	onlyOwner
_setMarketFlashLoanCaps	External	Can Modify State	onlyOwner
_setMarketSupplyCaps	External	Can Modify State	onlyOwner
_setCreditLimit	Public	Can Modify State	-
_setCompToken	Public	Can Modify State	onlyOwner
_setSafetyVault	Public	Can Modify State	onlyOwner
_setSafetyVaultRatio	Public	Can Modify State	onlyOwner
_setCompSpeedGuardianPaused	Public	Can Modify State	onlyOwner
_setPendingSafetyGuardian	External	Can Modify State	-
_acceptSafetyGuardian	External	Can Modify State	-
getCreditLimit	External	-	-

QsConfig			
getBorrowCap	External	-	-
getSupplyCap	External	-	-
getFlashLoanCap	External	-	-
calculateSeizeTokenAllocation	Public	-	-
getCompAllocation	Public	-	-
getFlashFee	External	-	-
_setCompRatio	Public	Can Modify State	onlyOwner
isBlocked	Public	-	-
_addToWhitelist	Public	Can Modify State	onlyOwner
_removeFromWhitelist	Public	Can Modify State	onlyOwner
_addToBlacklist	Public	Can Modify State	onlyOwner
_removeFromBlacklist	Public	Can Modify State	onlyOwner
_setFlashLoanFeeRatio	Public	Can Modify State	onlyOwner
isContract	Internal	-	-

QsMdxLPDelegate			
Function Name	Visibility	Mutability	Modifiers
_becomeImplementation	Public	Can Modify State	-
claimMdx	Public	Can Modify State	-
borrow	External	Can Modify State	-

QsMdxLPDelegate			
repayBorrow	External	Can Modify State	-
repayBorrowBehalf	External	Can Modify State	-
liquidateBorrow	External	Can Modify State	-
transferTokens	Internal	Can Modify State	-
getCashPrior	Internal	-	-
doTransferIn	Internal	Can Modify State	-
doTransferOut	Internal	Can Modify State	-
seizeInternal	Internal	Can Modify State	-
redeem	External	Can Modify State	-
redeemUnderlying	External	Can Modify State	-
claimAndStakeMdx	Internal	Can Modify State	-
harvestComp	Internal	Can Modify State	-
updateLPSupplyIndex	Internal	Can Modify State	-
updateSupplierIndex	Internal	Can Modify State	-
mdxBalance	Internal	-	-
fTokenBalance	Internal	-	-
compBalance	Internal	-	-

QsMdxLPOracle			
Function Name	Visibility	Mutability	Modifiers

QsMdxLPOracle			
<Constructor>	Public	Can Modify State	-
decimals	External	-	-
description	External	-	-
version	External	-	-
getRoundData	Public	-	-
latestRoundData	External	-	-
getTokenPrice	Private	-	-
setChainlinkSource	External	Can Modify State	onlyOwner
sqrt	Internal	-	-

QsPriceOracleV3			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getUnderlyingPrice	Public	-	-
setUnderlyingPrice	Public	Can Modify State	onlyPriceAdmin
isValidPrice	Public	-	-
getChainlinkPrice	Public	-	-
setDirectPrice	Public	Can Modify State	onlyPriceAdmin whenNotPaused
setDirectPrice	Public	Can Modify State	onlyPriceAdmin whenNotPaused
setPrice	Private	Can Modify State	onlyPriceAdmin

QsPriceOracleV3			
setDirectPriceWithForce	Public	Can Modify State	onlyPriceAdmin
assetPrices	External	-	-
getPriceInfo	External	-	-
addPriceAdmin	Public	Can Modify State	onlyGovernance
removePriceAdmin	Public	Can Modify State	onlyGovernance
setPaused	Public	Can Modify State	onlyGovernance
setErrorHappened	Public	Can Modify State	onlyGovernance
transferGovernance	Public	Can Modify State	onlyGovernance

Qstroller			
Function Name	Visibility	Mutability	Modifiers
_setQsConfig	Public	Can Modify State	-
_setCompSpeeds	Public	Can Modify State	-
getCompAddress	Public	-	-
calculateSeizeTokenAllocation	Public	-	-
transferComp	Internal	Can Modify State	-
borrowAllowed	External	Can Modify State	-
flashLoanAllowed	External	-	-
getFlashLoanCap	External	-	-
mintAllowed	External	Can Modify State	-

Qstroller			
updateCompSupplyIndex	Internal	Can Modify State	-
updateCompBorrowIndex	Internal	Can Modify State	-
getHypotheticalAccountLiquidityInternal	Internal	-	-
liquidateBorrowAllowed	Public	Can Modify State	-
seizeAllowed	Public	Can Modify State	-
repayBorrowAllowed	Public	Can Modify State	-
_supportMarket	External	Can Modify State	-
_setPriceOracle	External	Can Modify State	-

SToken			
Function Name	Visibility	Mutability	Modifiers
seizeInternal	Internal	Can Modify State	-
isNativeToken	Public	-	-
maxFlashLoan	External	-	-
flashFee	External	-	-
getFlashFeeInternal	Internal	-	-
flashLoan	External	Can Modify State	-

4.3 Vulnerability Summary

[N1] [Suggestion] Native token receiving issue

Category: Others

Content

The fallback function is defined in the CEther contract to receive native tokens, but the mintInternal logic is not actually triggered.

Code location: contracts/compound/CEther.sol

```
    function () external payable {
    }
```

Solution

It is recommended to call the mint function for the user in the fallback function.

Status

Confirmed

[N2] [Suggestion] Missing event record

Category: Others

Content

In the QsConfig contract, the owner can modify the compSpeedGuardianPaused parameter through the _setCompSpeedGuardianPaused function. SafetyGuardian can modify the pendingSafetyGuardian parameter through the _setPendingSafetyGuardian function. The pendingSafetyGuardian role can receive safetyGuardian permissions through the _acceptSafetyGuardian function. However, no event recording was made.

Code location: contracts/QsConfig.sol

```
    function _setCompSpeedGuardianPaused(bool state) public onlyOwner returns (bool)
{
        compSpeedGuardianPaused = state;
        return state;
    }

    function _setPendingSafetyGuardian(address newPendingSafetyGuardian) external {
```

```
        require(msg.sender == safetyGuardian, "!safetyGuardian");

        pendingSafetyGuardian = newPendingSafetyGuardian;
    }

    function _acceptSafetyGuardian() external {
        require(msg.sender == pendingSafetyGuardian, "!pendingSafetyGuardian");

        safetyGuardian = pendingSafetyGuardian;
        pendingSafetyGuardian = address(0x0);
    }
```

Solution

It is recommended to record incidents when modifying sensitive parameters for follow-up self-examination or community review.

Status

Fixed

[N3] [Suggestion] Code redundancy issue

Category: Others

Content

In the QsConfig contract, getFlashFee is used to obtain flashFee, but the token parameter passed in from outside is not used.

In the SToken contract, the maxFlashLoan function is used to obtain flashLoanCap, but the token parameter passed in from outside is not used.

Code location:

contracts/QsConfig.sol

```
    function getFlashFee(address borrower, address token, uint256 amount) external
view returns (uint flashFee) {
        if (whitelist[borrower]) {
            return 0;
        }
```

```
        Exp memory flashLoanFeeRatioExp = Exp({mantissa:flashLoanFeeRatio});
        (, flashFee) = mulScalarTruncate(flashLoanFeeRatioExp, amount);

        token;
    }
```

contract/SToken.sol

```
    function maxFlashLoan(address token) external view returns (uint256) {
        token;
        return Qstroller(address(comptroller)).getFlashLoanCap(address(this));
    }
```

Solution

It is recommended to remove unused code.

Status

Confirmed

[N4] [Critical] Flashloan issue

Category: Design Logic Audit

Content

The lightning loan function is implemented in the SToken contract. It can directly lend the funds in the cToken, and obtain the amount of funds in the cToken through the getCashPrior function to check before and after borrowing. If a malicious user borrows funds through a flash loan and mortgages it to the current cToken contract, the check on getCashPrior will be bypassed. At this time, the user has returned the flash loan and made a deposit in the agreement.

Code location: contracts/SToken.sol

```
    function flashLoan(IERC3156FlashBorrower receiver, address token, uint256 amount,
  bytes calldata data) external returns (bool) {
        require(accrueInterest() == uint(Error.NO_ERROR), "Accrue interest failed");
```

```
        uint cashBefore = getCashPrior();
        require(cashBefore >= amount, "Insufficient liquidity");
        // 1. calculate fee
        uint fee = getFlashFeeInternal(token, amount);
        // 2. transfer fund  to receiver
        doTransferOut(address(uint160(address(receiver))), amount);
        // 3. update totalBorrows
        totalBorrows = add_(totalBorrows, amount);
        // 4. execute receiver's callback function
        receiver.onFlashLoan(msg.sender, token, amount, fee, data);
        // 5. check cash balance
        uint cashAfter = getCashPrior();
        require(cashAfter >= add_(cashBefore, fee), "Inconsistent balance");

        (MathError err, uint reservesFee)= mulScalarTruncate(Exp({mantissa:
reserveFactorMantissa}), fee);
        require(err == MathError.NO_ERROR, "Error to calculate flashloan reserve
fee");
        totalReserves = add_(totalReserves, reservesFee);
        totalBorrows = sub_(totalBorrows, amount);
        return true;
    }
```

Solution

It is recommended to separate the flashloan pool from the cToken pool.

Status

Fixed

[N5] [Suggestion] Potential calculation flaws in flashloan fees

Category: Design Logic Audit

Content

The flash loan function is implemented in the SToken contract, which will obtain the flash loan fee through the getFlashFeeInternal function according to the token parameters passed in by the user. If the token data passed by the user is trusted to obtain the cost, then the malicious user can control the passed token parameters to control the cost to be paid.

Code location: contracts/SToken.sol

```
    function flashLoan(IERC3156FlashBorrower receiver, address token, uint256 amount,
bytes calldata data) external returns (bool) {
        require(accrueInterest() == uint(Error.NO_ERROR), "Accrue interest failed");

        uint cashBefore = getCashPrior();
        require(cashBefore >= amount, "Insufficient liquidity");
        // 1. calculate fee
        uint fee = getFlashFeeInternal(token, amount);
        // 2. transfer fund  to receiver
        doTransferOut(address(uint160(address(receiver))), amount);
        // 3. update totalBorrows
        totalBorrows = add_(totalBorrows, amount);
        // 4. execute receiver's callback function
        receiver.onFlashLoan(msg.sender, token, amount, fee, data);
        // 5. check cash balance
        uint cashAfter = getCashPrior();
        require(cashAfter >= add_(cashBefore, fee), "Inconsistent balance");

        (MathError err, uint reservesFee)= mulScalarTruncate(Exp({mantissa:
reserveFactorMantissa}), fee);
        require(err == MathError.NO_ERROR, "Error to calculate flashloan reserve
fee");
        totalReserves = add_(totalReserves, reservesFee);
        totalBorrows = sub_(totalBorrows, amount);
        return true;
    }
```

Solution

It is recommended that the tokens paid for the fee be consistent with the source of the loaned tokens.

Status

Confirmed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002110260001	SlowMist Security Team	2021.10.15 - 2021.10.26	Passed

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 critical risk, 4 suggestion vulnerabilities. And 3 suggestion vulnerabilities were confirmed and being fixed; All other findings were fixed. The code was not deployed to the mainnet.

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 based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. 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 the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist