Smart Contract Security Audit Report

[2021]

Table Of Contents

1 Executive Summary

On 2021.11.25, the SlowMist security team received the team's security audit application for Tectonic, 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 Audit
- Design Logic Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

Audit version:

tectonic-contracts-20211122.zip:

5853c18d819cb3fb8648d79a8f46f5c2b610e1f93da16066a97fe1c3bcad5182

Fixed version:

tectonic-contracts-20211215.zip:

24b4da807fd20fd52ee1c38296a4c5cbb838042378877685404e49e18ddc5f58

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Risk of excessive authority	Authority Control Vulnerability	Medium	Confirmed
N2	Redundant code	Others	Suggestion	Ignored
N3	Risk of replay attack	Replay Vulnerability	Suggestion	Ignored
N4	Hard coded issue	Others	Suggestion	Fixed
N5	TonicSpeed bug	Design Logic Audit	Suggestion	Fixed

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:

TectonicLens			
Function Name	Visibility	Mutability	Modifiers
tTokenMetadata	Public	Can Modify State	-
tTokenMetadataAll	External	Can Modify State	-
tTokenBalances	Public	Can Modify State	-

TectonicLens			
tTokenBalancesAll	External	Can Modify State	-
tTokenUnderlyingPrice	Public	Can Modify State	-
tTokenUnderlyingPriceAll	External	Can Modify State	-
getAccountLimits	Public	Can Modify State	-
getGovReceipts	Public	-	-
getGovBravoReceipts	Public	-	-
setProposal	Internal	-	-
getGovProposals	External	-	-
setBravoProposal	Internal	-	-
getGovBravoProposals	External	-	-
getTonicBalanceMetadata	External	-	-
getTonicBalanceMetadataExt	External	Can Modify State	-
getTonicVotes	External	-	-
compareStrings	Internal	-	-
add	Internal	-	-
sub	Internal	-	-

TectonicCoreErrorReporter			
Function Name	Visibility	Mutability	Modifiers
fail	Internal	Can Modify State	-

TectonicCoreErrorReporter			
failOpaque	Internal	Can Modify State	-

TokenErrorReporter			
Function Name	Visibility	Mutability	Modifiers
fail	Internal	Can Modify State	-
failOpaque	Internal	Can Modify State	-

Exponential			
Function Name	Visibility	Mutability	Modifiers
getExp	Internal	-	-
addExp	Internal	-	-
subExp	Internal	-	-
mulScalar	Internal	-	-
mulScalarTruncate	Internal	-	-
mulScalarTruncateAddUInt	Internal	-	-
divScalar	Internal	-	-
divScalarByExp	Internal	-	-
divScalarByExpTruncate	Internal	-	-
mulExp	Internal	-	-
mulExp	Internal	-	-
mulExp3	Internal	-	-

Exponential			
divExp	Internal	-	-

ExponentialNoError			
Function Name	Visibility	Mutability	Modifiers
truncate	Internal	-	-
mul_ScalarTruncate	Internal	-	-
mul_ScalarTruncateAddUInt	Internal	-	-
lessThanExp	Internal	-	-
lessThanOrEqualExp	Internal	-	-
greaterThanExp	Internal	-	-
isZeroExp	Internal	-	-
safe224	Internal	-	-
safe32	Internal	-	-
add_	Internal	-	-
add_	Internal	-	-
add_	Internal	-	-
add_	Internal	-	-
sub_	Internal	-	-
sub_	Internal	-	-
sub_	Internal	-	-

ExponentialNoError			
sub_	Internal	-	-
mul_	Internal	-	-
mul_	Internal	-	-
mul_	Internal	-	-
mul_	Internal	-	-
mul_	Internal	-	-
mul_	Internal	-	-
mul_	Internal	-	-
mul_	Internal	-	-
div_	Internal	-	-
div_	Internal	-	-
div_	Internal	-	-
div_	Internal	-	-
div_	Internal	-	-
div_	Internal	-	-
div_	Internal	-	-
div_	Internal	-	-
fraction	Internal	-	-

Maximillion

Maximillion			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
repayBehalf	Public	Payable	-
repayBehalfExplicit	Public	Payable	-

TEther			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
mint	External	Payable	-
redeem	External	Can Modify State	-
redeemUnderlying	External	Can Modify State	-
borrow	External	Can Modify State	-
repayBorrow	External	Payable	-
repayBorrowBehalf	External	Payable	-
liquidateBorrow	External	Payable	-
_addReserves	External	Payable	-
<Fallback>	External	Payable	-
getCashPrior	Internal	-	-
doTransferIn	Internal	Can Modify State	-
doTransferOut	Internal	Can Modify State	-

TEther			
requireNoError	Internal	-	-

TToken			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	-
transferTokens	Internal	Can Modify State	-
transfer	External	Can Modify State	nonReentrant
transferFrom	External	Can Modify State	nonReentrant
approve	External	Can Modify State	-
allowance	External	-	-
balanceOf	External	-	-
balanceOfUnderlying	External	Can Modify State	-
getAccountSnapshot	External	-	-
getBlockNumber	Internal	-	-
borrowRatePerBlock	External	-	-
supplyRatePerBlock	External	-	-
totalBorrowsCurrent	External	Can Modify State	nonReentrant
borrowBalanceCurrent	External	Can Modify State	nonReentrant
borrowBalanceStored	Public	-	-
borrowBalanceStoredInternal	Internal	-	-

TToken			
exchangeRateCurrent	Public	Can Modify State	nonReentrant
exchangeRateStored	Public	-	-
exchangeRateStoredInternal	Internal	-	-
getCash	External	-	-
accrueInterest	Public	Can Modify State	-
mintInternal	Internal	Can Modify State	nonReentrant
mintFresh	Internal	Can Modify State	-
redeemInternal	Internal	Can Modify State	nonReentrant
redeemUnderlyingInternal	Internal	Can Modify State	nonReentrant
redeemFresh	Internal	Can Modify State	-
borrowInternal	Internal	Can Modify State	nonReentrant
borrowFresh	Internal	Can Modify State	-
repayBorrowInternal	Internal	Can Modify State	nonReentrant
repayBorrowBehalfInternal	Internal	Can Modify State	nonReentrant
repayBorrowFresh	Internal	Can Modify State	-
liquidateBorrowInternal	Internal	Can Modify State	nonReentrant
liquidateBorrowFresh	Internal	Can Modify State	-
seize	External	Can Modify State	nonReentrant
seizeInternal	Internal	Can Modify State	-
_setPendingAdmin	External	Can Modify State	-

TToken			
_acceptAdmin	External	Can Modify State	-
_setTectonicCore	Public	Can Modify State	-
_setReserveFactor	External	Can Modify State	nonReentrant
_setReserveFactorFresh	Internal	Can Modify State	-
_addReservesInternal	Internal	Can Modify State	nonReentrant
_addReservesFresh	Internal	Can Modify State	-
_reduceReserves	External	Can Modify State	nonReentrant
_reduceReservesFresh	Internal	Can Modify State	-
_setInterestRateModel	Public	Can Modify State	-
_setInterestRateModelFresh	Internal	Can Modify State	-
getCashPrior	Internal	-	-
doTransferIn	Internal	Can Modify State	-
doTransferOut	Internal	Can Modify State	-

TectonicCoreInterface			
Function Name	Visibility	Mutability	Modifiers
enterMarkets	External	Can Modify State	-
exitMarket	External	Can Modify State	-
mintAllowed	External	Can Modify State	-
mintVerify	External	Can Modify State	-

TectonicCoreInterface			
redeemAllowed	External	Can Modify State	-
redeemVerify	External	Can Modify State	-
borrowAllowed	External	Can Modify State	-
borrowVerify	External	Can Modify State	-
repayBorrowAllowed	External	Can Modify State	-
repayBorrowVerify	External	Can Modify State	-
liquidateBorrowAllowed	External	Can Modify State	-
liquidateBorrowVerify	External	Can Modify State	-
seizeAllowed	External	Can Modify State	-
seizeVerify	External	Can Modify State	-
transferAllowed	External	Can Modify State	-
transferVerify	External	Can Modify State	-
liquidateCalculateSeizeTokens	External	-	-

CarefulMath			
Function Name	Visibility	Mutability	Modifiers
mulUInt	Internal	-	-
divUInt	Internal	-	-
subUInt	Internal	-	-
addUInt	Internal	-	-

CarefulMath			
addThenSubUInt	Internal	-	-

TTokenInterface			
Function Name	Visibility	Mutability	Modifiers
transfer	External	Can Modify State	-
transferFrom	External	Can Modify State	-
approve	External	Can Modify State	-
allowance	External	-	-
balanceOf	External	-	-
balanceOfUnderlying	External	Can Modify State	-
getAccountSnapshot	External	-	-
borrowRatePerBlock	External	-	-
supplyRatePerBlock	External	-	-
totalBorrowsCurrent	External	Can Modify State	-
borrowBalanceCurrent	External	Can Modify State	-
borrowBalanceStored	Public	-	-
exchangeRateCurrent	Public	Can Modify State	-
exchangeRateStored	Public	-	-
getCash	External	-	-
accrueInterest	Public	Can Modify State	-

TTokenInterface			
seize	External	Can Modify State	-
_setPendingAdmin	External	Can Modify State	-
_acceptAdmin	External	Can Modify State	-
_setTectonicCore	Public	Can Modify State	-
_setReserveFactor	External	Can Modify State	-
_reduceReserves	External	Can Modify State	-
_setInterestRateModel	Public	Can Modify State	-

TErc20Interface			
Function Name	Visibility	Mutability	Modifiers
mint	External	Can Modify State	-
redeem	External	Can Modify State	-
redeemUnderlying	External	Can Modify State	-
borrow	External	Can Modify State	-
repayBorrow	External	Can Modify State	-
repayBorrowBehalf	External	Can Modify State	-
liquidateBorrow	External	Can Modify State	-
sweepToken	External	Can Modify State	-
_addReserves	External	Can Modify State	-

Reservoir

Reservoir			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
drip	Public	Can Modify State	-
add	Internal	-	-
sub	Internal	-	-
mul	Internal	-	-
min	Internal	-	-

PriceOracle			
Function Name	Visibility	Mutability	Modifiers
getUnderlyingPrice	External	-	-

Timelock			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Fallback>	External	Payable	-
setDelay	Public	Can Modify State	-
acceptAdmin	Public	Can Modify State	-
setPendingAdmin	Public	Can Modify State	-
queueTransaction	Public	Can Modify State	-
cancelTransaction	Public	Can Modify State	-

Timelock			
executeTransaction	Public	Payable	-
getBlockTimestamp	Internal	-	-

TErc20			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	-
mint	External	Can Modify State	-
redeem	External	Can Modify State	-
redeemUnderlying	External	Can Modify State	-
borrow	External	Can Modify State	-
repayBorrow	External	Can Modify State	-
repayBorrowBehalf	External	Can Modify State	-
liquidateBorrow	External	Can Modify State	-
sweepToken	External	Can Modify State	-
_addReserves	External	Can Modify State	-
getCashPrior	Internal	-	-
doTransferIn	Internal	Can Modify State	-
doTransferOut	Internal	Can Modify State	-
_delegateTonicLikeTo	External	Can Modify State	-

GovernorAlpha

GovernorAlpha			
Function Name	Visibility	Mutability	Modifiers
quorumVotes	Public	-	-
proposalThreshold	Public	-	-
proposalMaxOperations	Public	-	-
votingDelay	Public	-	-
votingPeriod	Public	-	-
<Constructor>	Public	Can Modify State	-
propose	Public	Can Modify State	-
queue	Public	Can Modify State	-
_queueOrRevert	Internal	Can Modify State	-
execute	Public	Payable	-
cancel	Public	Can Modify State	-
getActions	Public	-	-
getReceipt	Public	-	-
state	Public	-	-
castVote	Public	Can Modify State	-
castVoteBySig	Public	Can Modify State	-
_castVote	Internal	Can Modify State	-
__acceptAdmin	Public	Can Modify State	-
__abdicate	Public	Can Modify State	-

GovernorAlpha			
__queueSetTimelockPendingAdmin	Public	Can Modify State	-
__executeSetTimelockPendingAdmin	Public	Can Modify State	-
add256	Internal	-	-
sub256	Internal	-	-
getChainId	Internal	-	-

GovernorBravoDelegate			
Function Name	Visibility	Mutability	Modifiers
initialize	Public	Can Modify State	-
propose	Public	Can Modify State	-
queue	External	Can Modify State	-
queueOrRevertInternal	Internal	Can Modify State	-
execute	External	Payable	-
cancel	External	Can Modify State	-
getActions	External	-	-
getReceipt	External	-	-
state	Public	-	-
castVote	External	Can Modify State	-
castVoteWithReason	External	Can Modify State	-
castVoteBySig	External	Can Modify State	-

GovernorBravoDelegate			
castVoteInternal	Internal	Can Modify State	-
_setVotingDelay	External	Can Modify State	-
_setVotingPeriod	External	Can Modify State	-
_setProposalThreshold	External	Can Modify State	-
_initiate	External	Can Modify State	-
_setPendingAdmin	External	Can Modify State	-
_acceptAdmin	External	Can Modify State	-
add256	Internal	-	-
sub256	Internal	-	-
getChainIdInternal	Internal	-	-

GovernorBravoDelegator			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_setImplementation	Public	Can Modify State	-
delegateTo	Internal	Can Modify State	-
<Fallback>	External	Payable	-

Tonic			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-

Tonic			
allowance	External	-	-
approve	External	Can Modify State	-
balanceOf	External	-	-
transfer	External	Can Modify State	-
transferFrom	External	Can Modify State	-
delegate	Public	Can Modify State	-
delegateBySig	Public	Can Modify State	-
getCurrentVotes	External	-	-
getPriorVotes	Public	-	-
_delegate	Internal	Can Modify State	-
_transferTokens	Internal	Can Modify State	-
_moveDelegates	Internal	Can Modify State	-
_writeCheckpoint	Internal	Can Modify State	-
safe32	Internal	-	-
safe128	Internal	-	-
add128	Internal	-	-
sub128	Internal	-	-
getChainId	Internal	-	-

BaseJumpRateModelV2

BaseJumpRateModelV2			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Internal	Can Modify State	-
updateJumpRateModel	External	Can Modify State	-
utilizationRate	Public	-	-
getBorrowRateInternal	Internal	-	-
getSupplyRate	Public	-	-
updateJumpRateModelInternal	Internal	Can Modify State	-

DAIInterestRateModelV3			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	JumpRateModelV2
updateJumpRateModel	External	Can Modify State	-
getSupplyRate	Public	-	-
dsrPerBlock	Public	-	-
poke	Public	Can Modify State	-

JumpRateModelV2			
Function Name	Visibility	Mutability	Modifiers
getBorrowRate	External	-	-
<Constructor>	Public	Can Modify State	BaseJumpRateModelV2

InterestRateModel			
Function Name	Visibility	Mutability	Modifiers
getBorrowRate	External	-	-
getSupplyRate	External	-	-

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

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

4.3 Vulnerability Summary

[N1] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability

Content

1. In the TectonicCore, TToken and TectonicOracleAdapter contracts, the admin role can modify key sensitive parameters such as the oracle price, the rate model, and the market, which will lead to the risk of excessive authority of the admin role.
2. In the SimplePriceOracle contract, the admin role can set the price of the underlying asset, which will lead to excessive authority.

Solution

It is recommended that the authority division processing be carried out in the early stage of the project:

1. The authority related to user funds should be transferred to the timelock contract or community governance. The timelock contract admin can use multi-signature to avoid the risk of private key leakage.
2. In the early stage of the project, some parameters may be frequently modified. This part of the permissions can be controlled separately.
3. Consider retaining the authority to temporarily suspend the project in order to respond to an emergency in the early stage of the project, which can quickly suspend the project and stop the loss in time.
4. After the project has passed the early stage of smooth operation, the authority can be transferred to community governance.

Status

Confirmed; 1. After communicating with the project team, they stated that the price oracle is initially using a fully centralized solution and the details of the centralized solution will be fully transparent to users (the centralized oracle soultion is not in the audit scope). Later, the oracle adapter will be updated and upgraded to a decentralized solution.

2. After communicating with the project team, they stated that the SimplePriceOracle is a test-only contract and will not be deployed.

3. After communicating with the project team, they stated that the admin role will be effective till the migration to a

token-holder governance model. This will be communicated over public channels. People will be aware of this decision and we will be transparent about this.

[N2] [Suggestion] Redundant code

Category: Others

Content

There are no specific logic implementations in the mintVerify, borrowVerify, repayBorrowVerify, liquidBorrowVerify, seizeVerify, and transferVerify, functions in the Comptroller contract, and they are called in the TToken contract.

Code location: contracts 2/TectonicCore.sol#L283-294, 425-435, 474-491, 547-566, 612-628, 665-676

```
    function mintVerify(address tToken, address minter, uint actualMintAmount, uint
mintTokens) external {
        // Shh - currently unused
        tToken;
        minter;
        actualMintAmount;
        mintTokens;

        // Shh - we don't ever want this hook to be marked pure
        if (false) {
            maxAssets = maxAssets;
        }
    }

    function borrowVerify(address tToken, address borrower, uint borrowAmount)
external {
        // Shh - currently unused
        tToken;
        borrower;
        borrowAmount;

        // Shh - we don't ever want this hook to be marked pure
        if (false) {
            maxAssets = maxAssets;
        }
    }

    function repayBorrowVerify(
```

```
        address tToken,
        address payer,
        address borrower,
        uint actualRepayAmount,
        uint borrowerIndex) external {
        // Shh - currently unused
        tToken;
        payer;
        borrower;
        actualRepayAmount;
        borrowerIndex;

        // Shh - we don't ever want this hook to be marked pure
        if (false) {
            maxAssets = maxAssets;
        }
    }

    function liquidateBorrowVerify(
        address tTokenBorrowed,
        address tTokenCollateral,
        address liquidator,
        address borrower,
        uint actualRepayAmount,
        uint seizeTokens) external {
        // Shh - currently unused
        tTokenBorrowed;
        tTokenCollateral;
        liquidator;
        borrower;
        actualRepayAmount;
        seizeTokens;

        // Shh - we don't ever want this hook to be marked pure
        if (false) {
            maxAssets = maxAssets;
        }
    }

    function seizeVerify(
        address tTokenCollateral,
        address tTokenBorrowed,
        address liquidator,
        address borrower,
```

```
        uint seizeTokens) external {
        // Shh - currently unused
        tTokenCollateral;
        tTokenBorrowed;
        liquidator;
        borrower;
        seizeTokens;

        // Shh - we don't ever want this hook to be marked pure
        if (false) {
            maxAssets = maxAssets;
        }
    }

    function transferVerify(address tToken, address src, address dst, uint
transferTokens) external {
        // Shh - currently unused
        tToken;
        src;
        dst;
        transferTokens;

        // Shh - we don't ever want this hook to be marked pure
        if (false) {
            maxAssets = maxAssets;
        }
    }
```

Solution

If subsequent business iterations do not need to use the above interface, it is recommended to remove these redundant functions.

Status

Ignored

[N3] [Suggestion] Risk of replay attack

Category: Replay Vulnerability

Content

The delegateBySig function uses the ecrecover of the contract to verify the signature. The signature data and v, r, s are passed in from the chain. If the random k value under the chain is repeated, the private key of the signature will be calculated.

Code loaction: contracts 2/Governance/Tonic.sol#L161-170

```
    function delegateBySig(address delegatee, uint nonce, uint expiry, uint8 v,
bytes32 r, bytes32 s) public {
        bytes32 domainSeparator = keccak256(abi.encode(DOMAIN_TYPEHASH,
keccak256(bytes(name)), getChainId(), address(this)));
        bytes32 structHash = keccak256(abi.encode(DELEGATION_TYPEHASH, delegatee,
nonce, expiry));
        bytes32 digest = keccak256(abi.encodePacked("\x19\x01", domainSeparator,
structHash));
        address signatory = ecrecover(digest, v, r, s);
        require(signatory != address(0), "Tonic::delegateBySig: invalid signature");
        require(nonce == nonces[signatory]++, "Tonic::delegateBySig: invalid nonce");
        require(now <= expiry, "Tonic::delegateBySig: signature expired");
        return _delegate(signatory, delegatee);
    }
```

Solution

It is recommended that when generating signature data off-chain, avoid repetition of random k values, resulting in corresponding repeated r values being submitted to the chain.

Referance: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-2612.md

Status

Ignored

[N4] [Suggestion] Hard coded issue

Category: Others

Content

In the TectonicCore contract, users can get the Tonic token address through the getTonicAddress function respectively, but the address is hard-coded directly into the contract, so if the address of the contract changes later,

it may affect the normal business function of the contract.

Code location: contracts 2/TectonicCore.sol#L1388-1390

```
    function getTonicAddress() public view returns (address) {
        return 0xc00e94Cb662C3520282E6f5717214004A7f26888;
    }
```

Solution

If the address of the above contract may change in the future, it is recommended to add an interface for changing the address.

Status

Fixed

[N5] [Suggestion] TonicSpeed bug

Category: Design Logic Audit

Content

In the TectonicCore contract, the setTonicSpeedInternal function, because the tonicSpeed of a Market can be set to zero (indicating the suspension of rewards). So there will be such a situation, when we first set the tonicSpeed of a market to zero, and then want to restart the reward after a period of time, then call setTonicSpeed function to set tonicSpeed to a non-zero value. At this time, it will enter the branch of else if (tonicSpeed != 0), which functions as an uninitialized market, initializing the reward index (index) and block number (block). However, even though the index of the suspended market is 0, the block will always be updated. Therefore, when we reset compSpeed for a suspended market: index will not be initialized, and at this time in the distributeSupplierTonic function, the user's reward index (supplierIndex) will be initialized to tonicInitialIndex (1e36), the supplyIndex of the market is 0 due to the above problem. This leads to underflow in

Double memory deltaIndex = sub_(supplyIndex=0,supplierIndex=1e36).

According to the description, there will be two issues:

1.borrow/supply state index not properly set when an active market gets its tonicSpeed set for the first time causing the market to get bricked.

2.extra interest is accrued in markets where their tonicSpeed is non-zero, set to zero, then set to non-zero again with time and market activity in-between.

Code location:

contracts 2/TectonicCore.sol#L1105-1136, 1189-1205

```
    function setTonicSpeedInternal(TToken tToken, uint tonicSpeed) internal {
        uint currentTonicSpeed = tonicSpeeds[address(tToken)];
        if (currentTonicSpeed != 0) {
            // note that TONIC speed could be set to 0 to halt liquidity rewards for
a market
            Exp memory borrowIndex = Exp({mantissa: tToken.borrowIndex()});
            updateTonicSupplyIndex(address(tToken));
            updateTonicBorrowIndex(address(tToken), borrowIndex);
        } else if (tonicSpeed != 0) {
            // Add the TONIC market
            Market storage market = markets[address(tToken)];
            require(market.isListed == true, "tonic market is not listed");

            if (tonicSupplyState[address(tToken)].index == 0 &&
tonicSupplyState[address(tToken)].block == 0) {
                tonicSupplyState[address(tToken)] = TonicMarketState({
                    index: tonicInitialIndex,
                    block: safe32(getBlockNumber(), "block number exceeds 32 bits")
                });
            }

            if (tonicBorrowState[address(tToken)].index == 0 &&
tonicBorrowState[address(tToken)].block == 0) {
                tonicBorrowState[address(tToken)] = TonicMarketState({
                    index: tonicInitialIndex,
                    block: safe32(getBlockNumber(), "block number exceeds 32 bits")
                });
            }
        }

        if (currentTonicSpeed != tonicSpeed) {
            tonicSpeeds[address(tToken)] = tonicSpeed;
```

```
            emit TonicSpeedUpdated(tToken, tonicSpeed);
        }
    }

    function distributeSupplierTonic(address tToken, address supplier) internal {
        TonicMarketState storage supplyState = tonicSupplyState[tToken];
        Double memory supplyIndex = Double({mantissa: supplyState.index});
        Double memory supplierIndex = Double({mantissa: tonicSupplierIndex[tToken]
[supplier]});
        tonicSupplierIndex[tToken][supplier] = supplyIndex.mantissa;

        if (supplierIndex.mantissa == 0 && supplyIndex.mantissa > 0) {
            supplierIndex.mantissa = tonicInitialIndex;
        }

        Double memory deltaIndex = sub_(supplyIndex, supplierIndex);
        uint supplierTokens = TToken(tToken).balanceOf(supplier);
        uint supplierDelta = mul_(supplierTokens, deltaIndex);
        uint supplierAccrued = add_(tonicAccrued[supplier], supplierDelta);
        tonicAccrued[supplier] = supplierAccrued;
        emit DistributedSupplierTonic(TToken(tToken), supplier, supplierDelta,
supplyIndex.mantissa);
    }
```

Solution

It's recommended to set the judgment of market.isListed before the judgment process of tonicSpeed.

Referance:

https://compound.finance/governance/proposals/62

https://www.comp.xyz/t/comptroller-compspeed-bug/2111

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002112080003	SlowMist Security Team	2021.11.25 - 2021.12.08	Medium Risk

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 medium risk, 4 suggestion vulnerabilities. And 1 medium risk vulnerabilities were confirmed and being fixed; 2 suggestion vulnerabilities were ignored; 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