CERTIK

Security Assessment

Meter.io-Sumer

May 12th, 2022

Table of Contents

About

Summary

This report has been prepared for Meter.io-Sumer to discover issues and vulnerabilities in the source code of the Meter.io-Sumer project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Meter.io-Sumer
Description	Meter.io-Sumer
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/HashPunk/Lending-Contract/tree/main/contracts
Commit	3091eef717b33a621992fb81fb6014d7d471ba75

Audit Summary

Delivery Date	May 12, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0	0
● Major	10	0	0	9	0	0	1
● Medium	3	0	0	2	0	0	1
● Minor	4	0	0	3	0	0	1
● Informational	5	0	0	1	0	1	3
● Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
ENE	ExponentialNoError.sol	418ae000ba621eb3e8ef0e4f2347310f0c2e5f3bb75b183681d8bf67c7c14b11
USB	UnderwriterStorage.sol	dde028ccd380609cf1e4ce32c3a775e56cc9dca34ca59bf302464e70b2325243
CEB	CErc20.sol	0d341e1b791797727737a44da7fa4633212543b86540ddaa6c498bb3877eccad
SUE	suErc20.sol	4c29fc2d2cbeee86149d21902d097342ece9e3696cc401a72fa84bcfdb0bd45d
CMB	CarefulMath.sol	dcb5b6857f6455d1daf77feb84a4cd11d3fb191fbc8097315479e88308f89083
CEI	CErc20Immutable.sol	6689cb8083354cf98dcfc49d00274046a205696451ab6df13ef3c28285c39052
RES	Reservoir.sol	b243c40d7ab525bf64435ef35dd5e283cbed0a3085ceb52205b5fa84ba94f3ab
CLL	Lens/CompoundLens.sol	51cb3b4080159336818917cf26c79e5d1ac05d36aa6da0cdff1d03d170a6c263
CDD	CDaiDelegate.sol	c98ee33d13672016db21d4d6353b45eccb5c9f77499df77c254574a0481c0c03
CGT	ComptrollerG4.sol.org	344bdfcd2db809dd044746ffc09ed7d24c389a736263f56250a19435254d5baa
UNI	Unitroller.sol	a56f8cf884f0bceb918bbb078aaa5cd3ef90002323787729d70fdee6b4a1c602
CGG	ComptrollerG6.sol.org	7399a584958cf6ccb30504dd2cbb3dcfb55bd841bc603f8158f55b588ba0ddae
DAI	DAIInterestRateModelV3.sol	5b7de4bd34a5cca672e22958ee2db42a25265a0b4bd9bfe0ccfd7b3f34d06b44
CSB	ComptrollerStorage.sol	a10b94f3c15f370087dd430c8880b4715be3200de101110d84150b2a20e32e7e
CTB	CToken.sol	0e4566df130b5c439fa1c67d6c249e79114b19a551e2d1da3aa900a0bf727b44
POB	PriceOracle.sol	8a5a574ee7b71ab417d5065cff4759ea32ce5c15f65e6e70fcbdd9a41d19c153
CGB	ComptrollerG1.sol.org	edd47b5300003c6bf4f61a5d34fd05188d968963ceb69dbb3666ce3605a7aa61
GBD	Governance/GovernorBravoDelegate.sol	551801cd444dcecac22a6ed5951aacb78bc6f597907a330573e5abd04b34a250

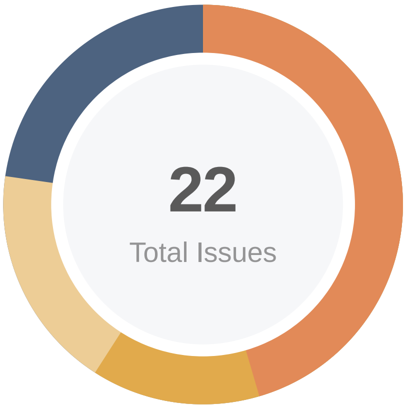
ID	File	SHA256 Checksum
SPO	SimplePriceOracle.sol	daebe63435b50a636f65496d286461820909a3bc895166c70c49f775554c465b
CED	CErc20Delegate.sol	9e4f5b92705c66f910bd0c38600bede344b592f1655a07e63a6ecfad45275a3b
BJR	BaseJumpRateModelV2.sol	32111c1b2bcdb051fa5c2564cd2a5e0662e699472ca5373499f67dca9c71cf47
CEU	CEther.sol	8ffbdc10a65ad384f12a91c07ff3a6579e0383ca7378b28f0263f432acba8778
JRV	JumpRateModelV2.sol	3c0a342bcce0fca28a0b460fc6a9c51c03eb4b3258c73140a14c0da8de242130
FPO	FeedPriceOracle.sol	b7200e156ae16b25bce72e31b0908c5359fc3de957410291594198f643c136ec
EIB	suErc20Immutable.sol	cfc00e00de33a16e0ee08b1cdf00c9f27d58ff83fd7b1e0924eb4a72545c3ec9
UWA	UnderWriterAdmin.sol	1faa45348c337ed2632d171c7a58f41d3fcaa6813eca27c27db46532b013730d
UPB	UnderwriterProxy.sol	f531428f08c1801b3da37d5785ee6dab486c41293600550f50fccbb1e1c32530
CGO	ComptrollerG7.sol.org	a1c6b1ed3e57d5899f4838dd6147413cdb48d7f4e9b1d378df4a9013602141c4
CGH	ComptrollerG3.sol.org	306dd7d02baa93d45d1bdf35909c785e9fbc943367b55fd047dcf4996009ba63
CTI	CTokenInterfaces.sol	7f4c7b71179dc6a859c4ebe1dc98dc40767fe8c050769d851bd6ede1a7c74d55
ERB	ErrorReporter.sol	a4eb51637fd29455d01f1b5b29bc0f1cb9a3b02f055e5aa8764932d8e8171f3c
JRM	JumpRateModel.sol	36a81d9c51869682d7428c80357b0bd5ce9c41abb5ca51015f115fe33ae3a0e1
TIM	Timelock.sol	ea4204fc8c5c72a5f4984177c209a16be5d538f1a3ee826744c901c21d27e382
CIB	ComptrollerInterface.sol	cb5865c24fbaf27a484b2d723172eede37694a4af38ff89a5c3447e22ad26170
IRM	InterestRateModel.sol	8bba52751bf2ca58e1d47012d0879a69d73e49c3de841bee79e3dfb5387b2433
WPI	WhitePaperInterestRateModel.sol	b5d06e0d725b01ecb8d0b88aa89300ddc0399904d84915a311f42f96970ba997
EDB	suErc20Delegate.sol	ad496ce10efb2800b41b25c4f423783176de40c5c7d4e9ad4fb0e51c2352b038
CGI	ComptrollerG5.sol.org	6ea55741e3ebc4eff82fba9a1bdf4d1609fcff9dee47cc89318816f528bceb04

ID	File	SHA256 Checksum
LJR	LegacyJumpRateModelV2.sol	99e34556232895653e5d87a456e13858e96f1856ad55ef1157c054dfd4260541
GBG	Governance/GovernorBravoDelegator.sol	489be8a9c67a544ed7538d1ffb5e53cd6440ef4c33ce40e1fa27d3e5f722b09c
CER	CErc20Delegator.sol	525e15dac623328c8c5cc9591be4fc7b5af85fdb96496ac3569201b63c26614b
LIR	LegacyInterestRateModel.sol	b6015e1f8ac5b818796beab7c14ccfb9aaee1f04d95216dd894c84c02d667a96
CGR	Governance/Comp.sol	2b557163c77b39edc8a4afedcc9ad0b5a25df65b0f3b1db6215bd8a47911b82b
CGU	ComptrollerG2.sol.org	5307859cd60d4a6bee5180798a7946cbfe0596a68e45d7ebe921efaf7f156680
EIP	EIP20Interface.sol	bc2ecd2927c202aab91222af287c07503cb348d8a96da3d368f195648356c4b7
SUT	suTokenInterestModel.sol	989a0fd12534ca50bf71ae2963b7267d6ca7e98d354ed850f61fad06fb6fcc8e
GAG	Governance/GovernorAlpha.sol	8a0553ad8bd250fc18710315dee64e3425550589c6466c01c3227fd8c7b3f1d4
COM	Comptroller.sol	f085e6988f93b1dec465419fd1dd3bc8fe734c0b39aa92da9a95fd0ab1b805f7
EIN	EIP20NonStandardInterface.sol	0994c25738db0bde158bc1d64ccd4ffd870ecf8780af6b267bf81aac04c11e4e
MAX	Maximillion.sol	32f9252032165bfe274fe16f0d74b3f7add6a037b7183dc964bcf01d0a5e687c
EDU	suErc20Delegator.sol	4295ca31489782421fbfdc6ca545d514f4cf30f7661799a562e8387b8fcaba70
EXP	Exponential.sol	35cd0b89d935713f89f679190d92764519f5afeb08accec6f813f6b7a0db5f4e
SMB	SafeMath.sol	204a19fb7a661c5bafcd5f7916254a457ca1fd9104e5708a73dd5010b11353dc
GBI	Governance/GovernorBravoInterfaces.sol	c095701d795af25ea725b1671cacfcecd690d76eb6ddfa1fd6d7de6bfffe7e81

Financial Models

Financial models of blockchain protocols need to be resilient to attacks. It needs to pass simulations and verifications to guarantee the security of the overall protocol. Financial models are not in the scope of the audit.

Findings



🟥 **Critical**	**0** (0.00%)
🟧 **Major**	**10** (45.45%)
🟨 **Medium**	**3** (13.64%)
🟨 **Minor**	**4** (18.18%)
🟦 **Informational**	**5** (22.73%)
🟩 **Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Centralization Related Risks	**Centralization / Privilege**	● **Major**	ⓘ Acknowledged
GLOBAL-02	Price Oracle Feed	**Data Flow, Centralization / Privilege**	● **Major**	ⓘ Acknowledged
GLOBAL-03	Third Party Dependencies	Volatile Code	● Minor	ⓘ Acknowledged
GLOBAL-04	Unlocked Compiler Version	Language Specific	● Informational	✓ Resolved
BJR-01	Centralization Related Risks	**Centralization / Privilege**	● **Major**	ⓘ Acknowledged
CGR-01	Initial Token Distribution	**Centralization / Privilege**	● **Medium**	ⓘ Acknowledged
COM-01	Centralization Related Risks	**Centralization / Privilege**	● **Major**	ⓘ Acknowledged
COM-02	Potential `mint/redeem/seize/transfer` Failure Possible	Logical Issue	● Minor	ⓘ Acknowledged
COM-03	Logical Issue Of The Function `getHypotheticalAccountLiquidityInternal()`	Logical Issue	● Minor	ⓘ Acknowledged
COM-04	Return Value Not Stored	Gas Optimization	● Informational	✓ Resolved

ID	Title	Category	Severity	Status
CON-01	Potential Anomal `exchangeRate` Risk Of The Function `sweepToken()`	Logical Issue	● Medium	Resolved
CON-02	Missing Zero Address Validation	Volatile Code	● Minor	Resolved
CON-03	Comparison To Boolean Constant	Coding Style	● Informational	Resolved
CON-04	Misuse Of Boolean Constant	Coding Style	● Informational	Acknowledged
CON-05	Declaration Naming Convention	Coding Style	● Informational	Partially Resolved
CTB-01	Checks-Effects-Interactions Pattern Violations	Logical Issue	● Major	Resolved
CTB-02	Logical Issue Of Function `exchangeRateStoredInternal()`	, Logical Issue	● Major	Acknowledged
CTB-03	Third Party Dependencies In The Contract `CToken`	Volatile Code	● Medium	Acknowledged
DAI-01	Centralization Related Risks	**Centralization / Privilege**	● **Major**	Acknowledged
FPO-01	Centralization Related Risks	**Centralization / Privilege**	● **Major**	Acknowledged
SUT-01	Centralization Related Risks	**Centralization / Privilege**	● **Major**	Acknowledged
UWA-01	Centralization Related Risks	**Centralization / Privilege**	● **Major**	Acknowledged

GLOBAL-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	● **Major**		ⓘ Acknowledged

Description

In the contracts `CToken/Unitroller/CErc20Delegator/GovernorBravoDelegator/CDaiDelegate`, the role `admin` has the authority over the following function:

- `_setComptroller()`: change the implementation of `Comptroller` with any contracts,
- `_setPendingImplementation()/_acceptImplementation()`: change the implementation of `Unitroller` with any contracts,
- `_setImplementation()`: change the implementation of `CErc20` with any contracts,
- `_setImplementation()`: change the implementation of `GovernorBravo` with any contracts,
- `_setPendingImplementation()/_acceptImplementation()`: change the implementation of the `UnderwriterAdmin` with any contracts,

Any compromise to the `admin` account may allow the hacker to take advantage of this and users' assets may suffer loss.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

The team acknowledged this issue and they will transfer the ownership to the multi-signature wallet in their own timeframe.

GLOBAL-02 | Price Oracle Feed

Category	Severity	Location	Status
Data Flow, Centralization / Privilege	● **Major**		ⓘ Acknowledged

Description

A serious issue was caused by Compound's centralized oracle solution which pulls market data from only a single exchange, Coinbase, with Uniswap TWAP used as a backstop.

Using Uniswap TWAP as a backstop is better than no backstop in this situation, but it introduces a false sense of security as it too can trivially be manipulated (as we saw during this event).

Recommendation

We recommend using the price oracle like Chainlink.

Alleviation

The team acknowledged this issue and they stated:

"They will use Chainlink or similar oracle service that uses various off-chain data sources in the deployment.

The price oracle feed in Sumer can be configured as "fixed price" or chainlink price feed or Uniswap.

Chainlink feeds will be considered with priority. They will only configure the alternatives unless the chainlink pair feed is unavailable."

GLOBAL-03 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	● Minor		ⓘ Acknowledged

Description

The contract is serving as the underlying entity to interact with third-party Chainlink, Witnet, SuToken protocols. The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts.

Recommendation

We understand that the business logic requires interaction with Chainlink, Witnet, SuToken, etc. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

The team acknowledged this issue and they will leave it as it is for now.

GLOBAL-04 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	● Informational		✓ Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to different compiler versions. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation

The team heeded our advice and resolved this issue in commit `809675068a80186ebf0561d96550c1ee275890c7`.

BJR-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	BaseJumpRateModelV2.sol: 66	ⓘ Acknowledged

Description

In the contract `BaseJumpRateModelV2` the role `owner` has authority over the following function:

- `updateJumpRateModel()`

Any compromise to the `owner` account may allow the hacker to take advantage of this authority.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

The team acknowledged this issue and they will transfer the ownership to the multi-signature wallet in their own timeframe.

CGR-01 | Initial Token Distribution

Category	Severity	Location	Status
Centralization / Privilege	● **Medium**	Governance/Comp.sol	ⓘ Acknowledged

Description

All of the `Comp` tokens are sent to the given address `account` when deploying the contract. This could be a centralization risk as the deployer can distribute all tokens without obtaining the consensus of the community.

Recommendation

We recommend the team to be transparent regarding the initial token distribution process, and the team shall make enough efforts to restrict the access of the private key.

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Alleviation

The team acknowledged this issue and they stated:

"This contract will not be used in production"

COM-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	Comptroller.sol	ⓘ Acknowledged

Description

In the contract `Comptroller` the role `admin` has authority over the following functions:

- `setMaxSupply()`
- `_setPriceOracle()`
- `_setCloseFactor()`
- `_setUnderWriterAdmin()`
- `_setLiquidationIncentive()`
- `_supportMarket()`
- `_grantComp()`
- `_setCompSpeeds()`
- `_setContributorCompSpeed()`

Any compromise to the `admin` account may allow the hacker to take advantage of this authority.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

The team acknowledged this issue and they will transfer the ownership to the multi-signature wallet in their own timeframe.

COM-02 | Potential `mint/redeem/seize/transfer` Failure Possible

Category	Severity	Location	Status
Logical Issue	● Minor	Comptroller.sol: 1601, 1649	ⓘ Acknowledged

Description

According to the codes in the function `distributeSupplierComp()`, the function is used to calculate the amount of Comp that needs to distribute to the supplier. The amount is calculated by the `deltaIndex`, which is calculated by the block-related parameters `supplyIndex(compSupplyState[cToken].index)` and `supplierIndex`. `supplierIndex` may be the value of `compInitialIndex`.

```
1649    function distributeSupplierComp(address cToken, address supplier) internal {
1650      // TODO: Don't distribute supplier COMP if the user is not in the supplier market.
1651      // This check should be as gas efficient as possible as distributeSupplierComp is called in many places.
1652      // - We really don't want to call an external contract as that's quite expensive.
1653
1654      CompMarketState storage supplyState = compSupplyState[cToken];
1655      uint256 supplyIndex = supplyState.index;
1656      uint256 supplierIndex = compSupplierIndex[cToken][supplier];
1657
1658      // Update supplier's index to the current index since we are distributing accrued COMP
1659      compSupplierIndex[cToken][supplier] = supplyIndex;
1660
1661      if (supplierIndex == 0 && supplyIndex >= compInitialIndex) {
1662        // Covers the case where users supplied tokens before the market's supply state index was set.
1663        // Rewards the user with COMP accrued from the start of when supplier rewards were first
1664        // set for the market.
1665        supplierIndex = compInitialIndex;
1666      }
1667
1668      // Calculate change in the cumulative sum of the COMP per cToken accrued
1669      Double memory deltaIndex = Double({mantissa: sub_(supplyIndex, supplierIndex)});
1670
1671      uint256 supplierTokens = CToken(cToken).balanceOf(supplier);
1672
1673      // Calculate COMP accrued: cTokenAmount * accruedPerCToken
1674      uint256 supplierDelta = mul_(supplierTokens, deltaIndex);
1675
```

```
  uint256 supplierAccrued = add_(compAccrued[supplier], supplierDelta);
  compAccrued[supplier] = supplierAccrued;

  emit DistributedSupplierComp(CToken(cToken), supplier, supplierDelta, supplyIndex);
}
```

According to the codes in the function `updateCompSupplyIndex()`, `compSupplyState[cToken].index` is calculated by the block and the `supplySpeed`, which may be smaller the value of `compInitialIndex` in case `compSupplyState[cToken]` is initialized incorrectly.

```
function updateCompSupplyIndex(address cToken) internal {
  CompMarketState storage supplyState = compSupplyState[cToken];
  uint256 supplySpeed = compSupplySpeeds[cToken];
  uint32 blockNumber = safe32(getBlockNumber(), 'block number exceeds 32 bits');
  uint256 deltaBlocks = sub_(uint256(blockNumber), uint256(supplyState.block));
  if (deltaBlocks > 0 && supplySpeed > 0) {
    uint256 supplyTokens = CToken(cToken).totalSupply();
    uint256 compAccrued = mul_(deltaBlocks, supplySpeed);
    Double memory ratio = supplyTokens > 0 ? fraction(compAccrued, supplyTokens) : Double({mantissa: 0});
    supplyState.index = safe224(
      add_(Double({mantissa: supplyState.index}), ratio).mantissa,
      'new index exceeds 224 bits'
    );
    supplyState.block = blockNumber;
  } else if (deltaBlocks > 0) {
    supplyState.block = blockNumber;
  }
}
```

As a result, the function `distributeSupplierComp()` called in the functions `mintAllowed()/redeemAllowed()/seizeAllowed()/transferAllowed()` will fail as subtraction overflow may be caused when calculating `deltaIndex`.

Recommendation

We recommend initializing the `compSupplyState[cToken]` correctly when deploying.

Alleviation

The team acknowledged this issue and they will leave it as it is for now.

COM-03 | Logical Issue Of The Function `getHypotheticalAccountLiquidityInternal()`

Category	Severity	Location	Status
Logical Issue	● Minor	Comptroller.sol: 998	ⓘ Acknowledged

Description

The function `getHypotheticalAccountLiquidityInternal()` is used to calculate what the account liquidity would be if the given amounts were redeemed/borrowed.

When looping all groups to calculate the `sumCollateral` and `sumBorrowPlusEffects`, the following logic will offset the collateral and the borrow, rather than add them separately to the final `sumCollateral` and `sumBorrowPlusEffects`.

```
      // pre-process group information
      if (groupVars[i].cTokenBalanceSum >= groupVars[i].suTokenBorrowSum) {
        groupVars[i].cTokenBalanceSum = groupVars[i].cTokenBalanceSum -
groupVars[i].suTokenBorrowSum;
        groupVars[i].suTokenBorrowSum = 0;
      } else {
        groupVars[i].suTokenBorrowSum = groupVars[i].suTokenBorrowSum -
groupVars[i].cTokenBalanceSum;
        groupVars[i].cTokenBalanceSum = 0;
      }
```

Recommendation

We recommend the team to state for the logic and design of this.

Alleviation

The team acknowledged this issue and they stated:

"This is required by algorithm. The cTokenBalanceSum (assets) and suTokenBorrowSum (liabilities) in the same group should be offset first then do the assets/liabilities calculation between groups.

The collateral logic for Sumer is that they divide assets into groups. The assets in the same asset group are supposed to be very similar to each other, for example, USDC and BUSD. Therefore the intra-group collateral rate could be much higher than the inter-group rates. In addition when minting suTokens with intra group collaterals, there is a different collateral rate as well (close to 1). The collateral matching engine will try maximize the collateral rates. For example, it will start with suToken minting collateral rate, then maximizing the intra collaterals with the liability and finaly the inter group collaterals.

The goal is to maximize the collateral utilization for the user deposit based on his outstanding liability."

COM-04 | Return Value Not Stored

Category	Severity	Location	Status
Gas Optimization	● Informational	Comptroller.sol	✓ Resolved

Description

The return value of an external call is not stored in a local or state variable.

Examples:

```
function _supportMarket(CToken cToken, uint8 groupId) external returns (uint256) {
  ...
  cToken.isCToken(); // Sanity check to make sure its really a CToken
  ...
}
```

Recommendation

We recommend adding “require” statement for isRToken:

```
require(cToken.isCToken(),”This is not a CToken contract!”);
```

Alleviation

The team heeded our advice and resolved this issue in commit `6103700518e2ac77e1e4977ab4c011de06e3ab65`.

CON-01 | Potential Anomal `exchangeRate` Risk Of The Function `sweepToken()`

Category	Severity	Location	Status
Logical Issue	● Medium	CErc20.sol: 128; suErc20.sol: 127	✓ Resolved

Description

The function `sweepToken()` is used to sweep the assets(exclude underlying asset) to the admin. The check in the function `sweepToken()` is as follows.

```
require(address(token) != underlying, 'CErc20::sweepToken: can not sweep underlying
token');
```

For the specificity of the underlying asset protocol, the above check may be invalid. For example, the `TUSD` token has a secondary entry simply forwards any calls to the primary contract. As a result, the underlying asset can be transferred to the admin.

For more, the total amount of the underlying asset in the contract is `totalCash`, which is used in the calculation of the `exchangeRate`. The `exchangeRate` becoming abnormal can lead to more serious risks.

Recommendation

We recommend adding the balance validation as follows.

```
function sweepToken(EIP20NonStandardInterface token) external {
    require(address(token) != underlying, 'CErc20::sweepToken: can not sweep underlying
token');

    uint256 underlyingBalanceBefore = underlying.balanceOf(address(this));

    uint256 balance = token.balanceOf(address(this));
    token.transfer(admin, balance);

    uint256 underlyingBalanceAfter = underlying.balanceOf(address(this));
    require(underlyingBalanceBefore == underlyingBalanceAfter);
}
```

Alleviation

The team heeded our advice and resolved this issue in commit `12594db7a0399cf1089ea557a46ce523ced2db2a`.

CON-02 | Missing Zero Address Validation

Category	Severity	Location	Status
Volatile Code	● Minor	CErc20.sol: 40; CErc20Delegator.sol: 68; CToken.sol: 1150; Comptroller.sol: 1308; FeedPriceOracle.sol: 39; Governance/GovernorBravoDelegate.sol: 344; Timelock.sol: 55, 99; UnderWriterAdmin.sol: 96, 180, 221; UnderwriterProxy.sol: 53, 121; Unitroller.sol: 46, 95; suErc20Delegator.sol: 68; suTokenInterestModel.sol: 36	✓ Resolved

Description

Addresses should be checked before assignment or external calls to make sure they are not zero addresses.

- `CErc20.initialize()`
- `CErc20Delegator._setImplementation()`
- `CToken._setPendingAdmin()`
- `Comptroller._setUnderWriterAdmin()`
- `FeedPriceOracle.changeOwner()`
- `GovernorBravoDelegate._setPendingAdmi()`
- `Timelock.setPendingAdmin()`
- `imelock.executeTransaction()`
- `UnderwriterAdmin.setGovTokenAddress()`
- `UnderwriterAdmin._setBorrowCapGuardian()`
- `UnderwriterAdmin._setPauseGuardian()`
- `UnderwriterProxy._setPendingAdmin()`
- `UnderwriterProxy._setPendingImplementation()`
- `Unitroller._setPendingImplementation()`
- `Unitroller._setPendingAdmin()`
- `suErc20Delegator._setImplementation()`
- `SuTokenRateModel.changeOwner()`

Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

The team heeded our advice and resolved this issue in commit `299c0c73e1ef139a7c060853d2abbb9739916ec4`.

CON-03 | Comparison To Boolean Constant

Category	Severity	Location	Status
Coding Style	● Informational	CToken.sol: 344, 688, 758; Comptroller.sol: 227, 1018, 1062, 1777, 1784, 1904~1908; Governance/GovernorAlpha.sol: 265; Governance/GovernorBravoDelegate.sol: 260; UnderWriterAdmin.sol: 111, 125, 138, 151	✓ Resolved

Description

Boolean constants can be used directly and do not need to be compared to true or false.

File: contracts/CToken.sol (Line 344, Function `CToken.exchangeRateStoredInternal`)

```
        if (isCToken != true) {
```

File: contracts/CToken.sol (Line 688, Function `CToken.redeemFresh`)

```
        if ((isCToken == true) && (getCashPrior() < vars.redeemAmount)) {
```

File: contracts/CToken.sol (Line 758, Function `CToken.borrowFresh`)

```
        if ((isCToken == true) && (getCashPrior() < borrowAmount)) {
```

File: contracts/Comptroller.sol (Line 1018, Function `Comptroller.getHypotheticalAccountLiquidityInternal`)

```
    if ((address(cTokenModify) != address(0)) && (cTokenModify.isCToken() == false)) {
```

File: contracts/Comptroller.sol (Line 1062, Function `Comptroller.getHypotheticalAccountLiquidityInternal`)

```
      if (asset.isCToken() == true) {
```

File: contracts/Comptroller.sol (Line 1777, Function `Comptroller.claimComp`)

```
        if (borrowers == true) {
```

File: contracts/Comptroller.sol (Line 1784, Function `Comptroller.claimComp`)

```
        if (suppliers == true) {
```

File: contracts/Comptroller.sol (Line 1904-1908, Function `Comptroller.isDeprecated`)

```
    return
      markets[address(cToken)].equalAssetGrouId == 0 &&
      //borrowGuardianPaused[address(cToken)] == true &&
      UnderwriterAdminInterface(underWriterAdmin)._getBorrowPaused(cToken) == true &&
      cToken.reserveFactorMantissa() == 1e18;
```

File: contracts/Comptroller.sol (Line 227, Function `Comptroller.addToMarketInternal`)

```
    if (marketToJoin.accountMembership[borrower] == true) {
```

File: contracts/Governance/GovernorAlpha.sol (Line 265, Function `GovernorAlpha._castVote`)

```
        require(receipt.hasVoted == false, "GovernorAlpha::_castVote: voter already
voted");
```

File: contracts/Governance/GovernorBravoDelegate.sol (Line 260, Function `GovernorBravoDelegate.castVoteInternal`)

```
        require(receipt.hasVoted == false, "GovernorBravo::castVoteInternal: voter
already voted");
```

File: contracts/UnderWriterAdmin.sol (Line 111, Function `UnderwriterAdmin._setMintPaused`)

```
    require(msg.sender == admin || state == true, 'only admin can unpause');
```

File: contracts/UnderWriterAdmin.sol (Line 125, Function `UnderwriterAdmin._setBorrowPaused`)

```
require(msg.sender == admin || state == true, 'only admin can unpause');
```

File: contracts/UnderWriterAdmin.sol (Line 138, Function `UnderwriterAdmin._setTransferPaused`)

```
require(msg.sender == admin || state == true, 'only admin can unpause');
```

File: contracts/UnderWriterAdmin.sol (Line 151, Function `UnderwriterAdmin._setSeizePaused`)

```
require(msg.sender == admin || state == true, 'only admin can unpause');
```

Recommendation

We recommend removing the equality to the boolean constant.

Alleviation

The team heeded our advice and resolved this issue in commit `cdfc9597b8854ed2f43c9631a3fa7195506af282`.

CON-04 | Misuse Of Boolean Constant

Category	Severity	Location	Status
Coding Style	● Informational	CErc20Delegate.sol: 25, 37; Comptroller.sol: 436, 604, 662, 744, 810, 868; suErc20Delegate.sol: 25, 37	ⓘ Acknowledged

Description

Boolean constants in code have only a few legitimate uses. Other uses (in complex expressions, as conditionals) indicate either an error or, most likely, the persistence of faulty code.

File: contracts/CErc20Delegate.sol (Line 25, Function `CErc20Delegate._becomeImplementation`)

```
        if (false) {
```

File: contracts/CErc20Delegate.sol (Line 37, Function `CErc20Delegate._resignImplementation`)

```
        if (false) {
```

File: contracts/Comptroller.sol (Line 604, Function `Comptroller.borrowVerify`)

```
    if (false) {
```

File: contracts/Comptroller.sol (Line 744, Function `Comptroller.liquidateBorrowVerify`)

```
    if (false) {
```

File: contracts/Comptroller.sol (Line 436, Function `Comptroller.mintVerify`)

```
    if (false) {
```

File: contracts/Comptroller.sol (Line 662, Function `Comptroller.repayBorrowVerify`)

```
    if (false) {
```

File: contracts/Comptroller.sol (Line 810, Function `Comptroller.seizeVerify`)

```
    if (false) {
```

File: contracts/Comptroller.sol (Line 868, Function `Comptroller.transferVerify`)

```
    if (false) {
```

File: contracts/suErc20Delegate.sol (Line 25, Function `suErc20Delegate._becomeImplementation`)

```
        if (false) {
```

File: contracts/suErc20Delegate.sol (Line 37, Function `suErc20Delegate._resignImplementation`)

```
        if (false) {
```

Recommendation

We recommend removing the ineffectual code.

Alleviation

The team heeded our advice and resolved this issue in commit `75d2908974dfdf19658200ee2db5411456198f7a`.

CON-05 | Declaration Naming Convention

Category	Severity	Location	Status
Coding Style	● Informational	BaseJumpRateModelV2.sol; ComptrollerInterface.sol; Comptroller.sol; DAIInterestRateModelV3.sol; ExponentialNoError.sol; Governance/GovernorAlpha.sol; Governance/GovernorBravoDelegate.sol; Governance/GovernorBravoInterfaces.sol; InterestRateModel.sol; JumpRateModel.sol; LegacyInterestRateModel.sol; PriceOracle.sol ; WhitePaperInterestRateModel.sol	Partially Resolved

Description

One or more declarations do not conform to the Solidity style guide with regards to its naming convention.

Particularly:

- `camelCase`: Should be applied to function names, argument names, local and state variable names, modifiers
- `UPPER_CASE`: Should be applied to `constant` variables
- `CapWords`: Should be applied to contract names, struct names, event names and enums

Examples:

Constants are not in `UPPER_CASE`:

- contract `BaseJumpRateModelV2`: `blocksPerYear`
- contract `CTokenInterfaces`: `protocolSeizeShareMantissa`, `borrowRateMaxMantissa`, `reserveFactorMaxMantissa`
- contract `Comptroller`: `compInitialIndex`, `closeFactorMinMantissa`, `closeFactorMaxMantissa`, `collateralFactorMaxMantissa`
- contract `ComptrollerInterface`: `isComptroller`
- contract `DAIInterestRateModelV3`: `assumedOneMinusReserveFactorMantissa`
- contract `ExponentialNoError`: `expScale`, `doubleScale`, `halfExpScale`, `mantissaOne`
- contract `GovernorBravoDelegate`: `quorumVotes`, `proposalMaxOperations`
- contract `InterestRateModel`: `isInterestRateModel`
- contract `JumpRateModel`: `blocksPerYear`
- contract `LegacyInterestRateModel`: `isInterestRateModel`
- contract `PriceOracle`: `isPriceOracle`

- contract `WhitePaperInterestRateModel`: `blocksPerYear`

Functions are not in `camelCase`

- contract `ExponentialNoError`: `mul_ScalarTruncate()`, `mul_ScalarTruncateAddUInt()`
- contract `GovernorAlpha`: `GRACE_PERIOD()`
- contract `GovernorBravoInterfaces`: `GRACE_PERIOD()`

Recommendation

We recommend adjusting those variable and function names to properly conform to Solidity's naming convention.

Alleviation

The team heeded our advice and partially resolved this issue in commit `75d2908974dfdf19658200ee2db5411456198f7a`.

CTB-01 | Checks-Effects-Interactions Pattern Violations

Category	Severity	Location	Status
Logical Issue	● Major	CToken.sol: 702, 794	✓ Resolved

Description

The following codes in the function `redeemFresh()/borrowFresh()` do not meet the Checks-Effects-Interactions pattern.

```
702          doTransferOut(redeemer, vars.redeemAmount);
703
704          /* We write previously calculated values into storage */
705          totalSupply = vars.totalSupplyNew;
706          accountTokens[redeemer] = vars.accountTokensNew;
```

```
794          doTransferOut(borrower, borrowAmount);
795
796          /* We write the previously calculated values into storage */
797          accountBorrows[borrower].principal = vars.accountBorrowsNew;
798          accountBorrows[borrower].interestIndex = borrowIndex;
799          totalBorrows = vars.totalBorrowsNew;
```

It only has a reentrancy lock as there is no lock at the controller level, only the CToken level.

If the `cToken` is an ERC777 protocol, the reentrancy can happen in function levels of an ERC777 based contract, i.e. multiple function calls that are triggered by the hook mechanism of ERC777.

This issue is possible to happen with all compound forks, but Compound is not affected as they do not list tokens with callback functionality.

Recommendation

We recommend using the Checks-Effects-Interactions pattern and understanding the security limitations of forking compound.

Alleviation

The team heeded our advice and resolved this issue in commit `798ad666780666eafd8f0ddae7339ee14c378258`.

CTB-02 | Logical Issue Of Function `exchangeRateStoredInternal()`

Category	Severity	Location	Status
, Logical Issue	● Major	CToken.sol: 342	ⓘ Acknowledged

Description

In the aforementioned line, the formula for the calculation of `exchangeRate` is as follows after cToken is minted:

$$exchangeRate = \frac{totalCash + totalBorrows - totalReserves}{totalSupply}$$

```
    function exchangeRateStoredInternal() internal view returns (MathError, uint) {

        if (isCToken != true) {
            return (MathError.NO_ERROR, initialExchangeRateMantissa);
        }

        uint _totalSupply = totalSupply;
        if (_totalSupply == 0) {
            /*
             * If there are no tokens minted:
             *  exchangeRate = initialExchangeRate
             */
            return (MathError.NO_ERROR, initialExchangeRateMantissa);
        } else {
            /*
             * Otherwise:
             *  exchangeRate = (totalCash + totalBorrows - totalReserves) / totalSupply
             */
            uint totalCash = getCashPrior();
            uint cashPlusBorrowsMinusReserves;
            Exp memory exchangeRate;
            MathError mathErr;

            (mathErr, cashPlusBorrowsMinusReserves) = addThenSubUInt(totalCash, totalBorrows, totalReserves);
            if (mathErr != MathError.NO_ERROR) {
                return (mathErr, 0);
            }

            (mathErr, exchangeRate) = getExp(cashPlusBorrowsMinusReserves, _totalSupply);
            if (mathErr != MathError.NO_ERROR) {
                return (mathErr, 0);
            }

            return (MathError.NO_ERROR, exchangeRate.mantissa);
        }
    }
```

In solidity, division calculations have truncation problems. The `totalSupply` will be 1 and `exchangeRate` will be much smaller than `initialExchangeRate` in case the last user redeems (`accountTokens[redeemer] - 1`) cToken.

As a result, the `exchangeRate` would be extremely small.

When the value of `exchangeRate` is much smaller than `initialExchangeRate`, the user can mint cTokens well above normal values, and then the value of `exchangeRate` will be normal with the interest generating. In other words, the users can use this arbitrage to take away the underlying tokens in this pool.

For example, the user can mint the amount of 1e8 CToken with one underlying token in case `exchangeRate = 1/1e8`.

Recommendation

We recommend using the following solutions to help mitigate this issue:

1. adding reasonable upper and lower boundaries to replace the return value when the `exchangeRate` is un-reasonable big or small,
2. adding a new contract that can only call `mint()` but can't call `redeem()` to supply reasonable amounts of the underlying token to the pool.

Alleviation

The team acknowledged this issue and they will leave it as it is for now.

CTB-03 | Third Party Dependencies In The Contract `CToken`

Category	Severity	Location	Status
Volatile Code	● Medium	CToken.sol	ⓘ Acknowledged

Description

The CToken contract is serving as the underlying entity to interact with third-party underlying asset protocols. The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that CToken's business logic requires interaction with the underlying asset protocol. We encourage the team to continuously monitor the status of third parties in order to mitigate side effects when unexpected activity is observed. The team should also identify if there are incompatibilities between the specificity of the underlying asset protocol and the combination of CToken and Comptroller contracts.

Alleviation

The team acknowledged this issue and they will take extreme caution when accepting new assets.

DAI-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	DAIInterestRateModelV3.sol: 51	ⓘ Acknowledged

Description

In the contract `DAIInterestRateModelV3` the role `owner` has authority over the following functions.

- `updateJumpRateModel()`

Any compromise to the `owner` account may allow the hacker to take advantage of this authority.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

The team acknowledged this issue and they will transfer the ownership to the multi-signature wallet in their own timeframe.

FPO-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	FeedPriceOracle.sol: 38, 43, 51, 65, 73, 77	ⓘ Acknowledged

Description

In the contract `FeedPriceOracle` the role `owner` has authority over the following functions.

- `setFeed()`
- `setWitnetFeed()`
- `removeFeed()`
- `setFixedPrice()`
- `removeFixedPrice(`)

Any compromise to the `owner` account may allow the hacker to take advantage of this authority.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

The team acknowledged this issue and they will transfer the ownership to the multi-signature wallet in their own timeframe.

SUT-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	suTokenInterestModel.sol	ⓘ Acknowledged

Description

In the contract `SuTokenRateModel` the role `owner` has authority over the following functions.

- `setBorrowRate()`
- `setSupplyRate()`

Any compromise to the `owner` account may allow the hacker to take advantage of this authority.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

The team acknowledged this issue and they will transfer the ownership to the multi-signature wallet in their own timeframe.

UWA-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	UnderWriterAdmin.sol: 108, 122, 136, 149, 177, 189, 214, 235	ⓘ Acknowledged

Description

In the contract `UnderwriterAdmin` the role `admin` has authority over the following functions.

- `_setBorrowCapGuardian()`
- `_setSuTokenRateMantissa()`
- `_setMintPaused()`
- `_setBorrowPaused()`
- `_setTransferPaused()`
- `_setSeizePaused()`
- `setGovTokenAddress()`
- `_setMarketBorrowCaps()`

Any compromise to the `admin` account may allow the hacker to take advantage of this authority.

In the contract `UnderwriterAdmin` the role `borrowCapGuardian` has authority over the following functions.

- `_setMarketBorrowCaps()`

Any compromise to the `borrowCapGuardian` account may allow the hacker to take advantage of this authority.

In the contract `UnderwriterAdmin` the role `pauseGuardian` has authority over the following functions.

- `_setMintPaused()`
- `_setBorrowPaused()`
- `_setTransferPaused()`
- `_setSeizePaused()`

Any compromise to the `pauseGuardian` account may allow the hacker to take advantage of this authority.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

The team acknowledged this issue and they will transfer the ownership to the multi-signature wallet in their own timeframe.

Appendix

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.

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.

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.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

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

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND

“AS AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER’S OR ANY OTHER PERSON’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK’S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER’S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED “AS IS” AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK’S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we’re able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.