Security Assessment

Justlend

Apr 8th, 2022

Table of Contents

Summary

This report has been prepared for Justlend to discover issues and vulnerabilities in the source code of the Justlend 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	Justlend
Description	Justlend
Platform	TRON
Language	Solidity
Codebase	https://github.com/justlend/justlend-protocol
Commit	bed296fc6205658cc5b5ca871f9ac9dd69e303d9

Audit Summary

Delivery Date	Apr 08, 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	6	0	0	1	2	1	2
Medium	1	0	0	1	0	0	0
Minor	2	0	0	2	0	0	0
Informational	7	0	0	7	0	0	0
Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
GAG	Governance/GovernorAlpha.sol	3436305f7b6350849565d4af975b6bfb4f9b9c69e58c26684bf8471e01f0d047
WJS	Governance/WJST.sol	01240dcb6aa705c2dd4594bb202f959deeba5f6eda98defc1f3863fe1452053e
CLL	Lens/CompoundLens.sol	3ae696d7bd8c3ff2cfd4afe7d8a2a031360d841635e6807ed0a9792e638781ca
DSV	PriceOracle/DSValue.sol	1cdb0eb5b51c9d85c9c8d01071df5c33c408ca40ae97ae63ebcd18776827ea19
POP	PriceOracle/PriceOracle.sol	9b258a3d95786c123c4904c9c53ac691c8b8e4e0c0a748a943a82e7cf4337fc7
BJR	BaseJumpRateModelV2.sol	bfcea7d2dcd937667063ea6631e14d6e95f3e0b598a49886936fd30e9bbb0e4d
CDD	CDaiDelegate.sol	e0dbb826eb0a0f6032cafce5da66259be8d65a14c5199f8b699c75348c516939
CEJ	CErc20.sol	6e0a3cae739460bb6025f0b4d9365925d4fab71dda9f49ff5f304d9432cb893f
CED	CErc20Delegate.sol	a41571abd99b06c298f121310d9dd51471d436630115e27a2489bfdc0487bd6c
CEC	CErc20Delegator.sol	4293707617b67003d9e9e332bd9032cf53d5b8840d37e1957658798df25ed214
CEI	CErc20Immutable.sol	4adb0a8b5cc3185d9b5382c54411fbd1d942bcec2052963ba952da35da4be515
CEK	CEther.sol	1b8e3c2ddfed8ce0899e9c95b3a330e3b5bf453eced781f15b7a3f123c5857c1

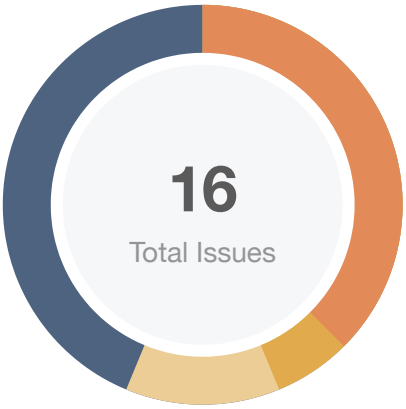
ID	File	SHA256 Checksum
CTJ	CToken.sol	873f6c80f20303a3ac8d3f44b06382875fe5e698f06f0921d45d99c649bc897e
CTI	CTokenInterfaces.sol	e606b5b73732c3da65a8815e8ed12e04ffe2e49c8543756f63fb697511ff98a3
CMJ	CarefulMath.sol	2d911545d6c21f79a82c1a6a508f226c3a1bc3d4eb5a483c774c6fe917cc7471
CJC	Comptroller.sol	888f089f77588430baf3b10155d1b28920c93ca69c51ae7aaeeee59dba2c90b0
CGJ	ComptrollerG1.sol	06e11a4fa5b68b62f9c6f1d4bbfc9a4ff77ea0b5f884832471ce6b755d026929
CGC	ComptrollerG2.sol	2d3812ae5ce1bf38be9420550d2380b387c3f739b36a9f90e89e27e181dbf39b
CIJ	ComptrollerInterface.sol	8db2ad9a3d7ded37bf0150e95d4f9c88578e089af5780daaac26b9257d3441ea
CSJ	ComptrollerStorage.sol	581b9a9eba36a3c22fce0d98bdd5d4057e5bb7db8f013ada048b1c1eb663eb2e
DAI	DAIInterestRateModelV2.sol	b1ef496f039beddbe7f9f5dda5b9b04f6deec8e1a198cc67b22d7cc6f7246cbb
EIP	EIP20Interface.sol	001f582fb5cc81d6b86a823d3a1c1b404d913508c0daee16783c5a0f52f88a6b
EIN	EIP20NonStandardInterface.sol	989255f54a70a25aba32604f35522dc0a86bfca96bfc1ec31e10cee01f3d458b
ERJ	ErrorReporter.sol	5f330db0d084e8dbb6d3e441e59f228d22e0d5081a6ca92a43b4bdb73d0d5a20
EJC	Exponential.sol	111fb77a19d9df5917e2383fe7241535bfa1bff43e23b566ff791d2ff1eb8b84

ID	File	SHA256 Checksum
IRM	InterestRateModel.sol	e4e42d3ecdbbfd059a74bfc30026bfd829e3c299633d73f376e67aba5ba9c8be
JRM	JumpRateModel.sol	8d63c94d135d8cc089fe5cf6c32da1ba96168bc27074797c835bce442145aca2
JRV	JumpRateModelV2.sol	442bc6845e48a9bfb4ef2c92824e0215ade8894e8932364c08d705c2557ae863
MJC	Maximillion.sol	fb427af3e7d5087267567ce9ea1b5bf1749c7c4b02d1a45df01d5815e837d84e
POC	PriceOracle.sol	8718a29632d6c196e7f9adeda407bb7431b6942d0b54db7f34c77984acd5eb0e
POK	PriceOracleProxy.sol	83be57183c145e4ba1f519e3ba9641a28356b00f6e07011e99af9772aeca3e6f
RJC	Reservoir.sol	ccc5b7daa6e94557ca9cb504405bf4201dbc2aaad1a95bb3b116f491b43bbbd8
SMJ	SafeMath.sol	cc214a7b44077774e2eb36e5141c551bfd00bb10e88828cc152b2dc585f8977e
SPO	SimplePriceOracle.sol	ec4b5176ae5872e876c54c1e34c3c66bc590d4c4c466b983a0a97fd5f85a39d3
TJC	Timelock.sol	720770bf02e476b74a9421837e21356a1cb3f50871331dbdf5c21538a488ab84
UJC	Unitroller.sol	5cc30091a4bcfc12775d1dfbb7561fa638a9f8ccde05faaa3d827fd8f1fe4392
WPI	WhitePaperInterestRateModel.sol	c53966c1409252916fc8a6f130ddca1e27f24aee87c7d7175d0fc58f21246658

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	**6** (37.50%)
Medium	**1** (6.25%)
Minor	**2** (12.50%)
Informational	**7** (43.75%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Centralization related risks	**Centralization / Privilege**	**Major**	Mitigated
GLOBAL-02	Price oracle feed	Data Flow	Medium	Acknowledged
GLOBAL-03	Missing input validation	Volatile Code	Minor	Acknowledged
GLOBAL-04	Unlocked compiler version declaration	Language Specific	Informational	Acknowledged
GLOBAL-05	Proper usage of “public” and “external” type	Coding Style	Informational	Acknowledged
GLOBAL-06	Incorrect naming convention utilization	Coding Style	Informational	Acknowledged
CJC-01	Misuse of a boolean constant	Coding Style	Informational	Acknowledged
CJC-02	Return value not stored	Gas Optimization	Informational	Acknowledged
CJC-03	Boolean equality	Gas Optimization	Informational	Acknowledged
CTJ-01	Checks-Effects-Interactions pattern violations	Logical Issue	Major	Resolved

ID	Title	Category	Severity	Status
CTJ-02	Logical issue of function `exchangeRateStoredInternal()`	Logical Issue	Major	Partially Resolved
GAG-01	Centralization related risks	**Centralization / Privilege**	**Major**	Resolved
POP-01	Centralization related risks	**Centralization / Privilege**	**Major**	Acknowledged
POP-02	Logical issue of `setPriceInternal()`	Control Flow	Minor	Acknowledged
WJS-01	Centralization related risks	**Centralization / Privilege**	**Major**	Mitigated
WJS-02	Vote for Multiple Active Proposals	Control Flow	Informational	Acknowledged

GLOBAL-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	**Major**	**Global**	Mitigated

Description

In the contracts `CToken/Unitroller/CErc20Delegator`, 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,

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., 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 the issue and adopted the Timelock solution to delay-sensitive operations at the current stage. The `CEther`, `Unitroller` and `CErc20Delegator` contracts have transferred the ownership to a Timelock contract with a minimal 48 hours delay. And the Timelock contract has transferred the ownership to a governance contract to increase transparency and user involvement.

The contracts are deployed at https://tronscan.org/#/contract/TE2RzoSV3wFK99w6J9UnnZ4vLfXYoxvRwP
https://tronscan.org/#/contract/TGjYzgCyPobsNS9n6WcbdLVR9dH7mWqFx7
https://tronscan.org/#/contract/TLjn59xNM7VEK6VZ3VQ8Y1ipxsdsFka5wZ
https://tronscan.org/#/contract/TXJgMdjVX5dKiQaUi9QobwNxtSQaFqccvd
https://tronscan.org/#/contract/TYSHTEq9NFSgst94saeRvt6rAYgWkqMFbj
https://tronscan.org/#/contract/TL5x9MtSnDy537FXKx53yAaHRRNdg9TkkA
https://tronscan.org/#/contract/TSCpzKvJfXHj1HW5jKg9dZA8z9aMxxGLd8
https://tronscan.org/#/contract/TGBr8uh9jBVHJhhkwSJvQN2ZAKzVkxDmno
https://tronscan.org/#/contract/TW3GyD3hYkKwzSGytWwWGXpe2a93zCpRzJ
https://tronscan.org/#/contract/TRg6MnpsFXc82ymUPgf5qbj59ibxiEDWvv
https://tronscan.org/#/contract/TVsKSRgRoMcCp798qqRGesXRfzy2MzRjkR
https://tronscan.org/#/contract/TLeEu311Cbw63BcmMHDgDLu7fnk9fqGcqT

https://tronscan.org/#/contract/TQ2sbnmxtR7jrNk4nxz2A8f9sneCqmk6SB
https://tronscan.org/#/contract/TWQhCXaWz4eHK4Kd1ErSDHjMFPoPc9czts
https://tronscan.org/#/contract/TV4WWBqBfn1kd4KmpYeSJpVAfybfrxEN9L
https://tronscan.org/#/contract/TUY54PVeH6WCcYCd6ZXXoBDsHytN9V5PXt
https://tronscan.org/#/contract/TLkUdtDBLMfJdXni2iTa4u2DKM53XmDJHi
https://tronscan.org/#/contract/TFpPyDCKvNFgos3g3WVsAqMrdqhB81JXHE
https://tronscan.org/#/contract/TPXDpkg9e3eZzxqxAUyke9S4z4pGJBJw9e
https://tronscan.org/#/contract/TM82erAZJSP7NKc17JdTnzVC8WKJHismWB
https://tronscan.org/#/contract/TSXv71Fy5XdL3Rh2QfBoUu3NAaM4sMif8R
https://tronscan.org/#/contract/THbrSjDsDA2KJRxx8K73tN7vLgaXSUNQFk
https://tronscan.org/#/contract/TNSBA6KvSvMoTqQcEgpVK7VhHT3z7wifxy
https://tronscan.org/#/contract/THQY8YX19jLFSFg1xhthM5wb7xZvKLCzgq
https://tronscan.org/#/contract/TR7BUFRQeq1w5jAZf1FKx85SHuX6PfMqsV
https://tronscan.org/#/contract/TQBvTVisiceDvsQVbLbcYyWQGWP7wtaQnc

The admin of contracts `CEther`, `Unitroller` and `CErc20Delegator` is a Timelock contract, which is deployed at `https://tronscan.org/#/contract/TRWNvb15NmfNKNLhQpxefFz7cNjrYjEw7x`.

The admin of contract `Timelock` is a governance contract, which is deployed at `https://tronscan.org/#/contract/TH1SVVVU9NF1ans3CRBCJ5kW2yvn4sHP9b`.

GLOBAL-02 | Price Oracle Feed

Category	Severity	Location	Status
Data Flow	Medium	Global	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 Chainlink as the price oracle.

Alleviation

The team acknowledged this issue and they stated:

"They will use the median feed of WinkLink, SunSwapV1, SunSwapV2, Binance, Coingecko, CoinMarketCap as the price source for price feed. The price can only be offset by ±10% at most within 30 minutes. They will use TWAP price as a backstop in the future."

GLOBAL-03 | Missing Input Validation

Category	Severity	Location	Status
Volatile Code	Minor	Global	Acknowledged

Description

The given input is missing the check for the non-zero address.

For example,

- contract `Comptroller`: `newPauseGuardian` in function `_setPauseGuardian()`,
- contract `CToken`: `newPendingAdmin` in function `_setPendingAdmin()`,
- contract `Unitroller`: `newPendingImplementation` in function `_setPendingImplementation()`, `newPendingAdmin` in function `_setPendingAdmin()`

Recommendation

We recommend adding the check for the passed-in values to prevent unexpected error.

Alleviation

The team acknowledged this issue and they stated:

"The impact of this problem is minimal. Given the contract has been deployed, it will not be modified for now.“

GLOBAL-04 | Unlocked Compiler Version Declaration

Category	Severity	Location	Status
Language Specific	Informational	Global	Acknowledged

Description

The compiler version utilized throughout the project uses the "^" prefix specifier, denoting that a compiler version which is greater than the version will be used to compile the contracts.

Recommendation

It is a general practice to alternatively lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and thus be able to identify emerging more easily. We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation

The team acknowledged this issue and they stated:

"The impact of this problem is minimal. Given the contract has been deployed, it will not be modified for now.“

GLOBAL-05 | Proper Usage Of “public” And “external” Type

Category	Severity	Location	Status
Coding Style	● Informational	Global	ⓘ Acknowledged

Description

“public” functions that are never called by the contract should be declared “external”. When the inputs are arrays, “external” functions are more efficient than “public” functions.

Examples:

Functions like :

- contract `Comptroller`: `enterMarkets()`, `getAccountLiquidity()`, `getHypotheticalAccountLiquidity()`, `_setPriceOracle()`, `_setPauseGuardian()`, `_setMintPaused()`, `_setBorrowPaused()`, `_setTransferPaused()`, `_setSeizePaused()`, `_become()`, `claimComp()`, `getAllMarkets()`,
- contract `CToken`: `initialize()`, `_setInterestRateModel()`,
- contract `Comp`: `delegate()`, `delegateBySig()`, `getPriorVotes()`
- contract `WJST`: `deposit()`, `withdraw()`, `getPriorVotes()`, `voteFresh()`, `withdrawVotes()`, `setGovernorAlpha()`, `transferOwnership()`,
- contract `Unitroller`: `_setPendingImplementation()`, `_setPendingAdmin()`, `_acceptAdmin()`,
- contract `CErc20`: `initialize()`,
- contract `CCErc20Delegate`: `_becomeImplementation()`, `_resignImplementation()`,
- contract `CErc20Delegator`: `borrowBalanceStored()`, `exchangeRateCurrent()`, `exchangeRateStored()`, `accrueInterest()`, `_setComptroller()`, `_setInterestRateModel()`,
- contract `GovernorAlpha`, `propose()`.

Recommendation

We recommend using the “external” attribute for functions never called from the contract.

Alleviation

The team acknowledged this issue and they stated:

"The impact of this problem is minimal. Given the contract has been deployed, it will not be modified for now.“

GLOBAL-06 | Incorrect Naming Convention Utilization

Category	Severity	Location	Status
Coding Style	● Informational	Global	ⓘ Acknowledged

Description

Solidity defines a naming convention that should be followed. In general, the following naming conventions should be utilized in a Solidity file:

Constants should be named with all capital letters with underscores separating words UPPER_CASE_WITH_UNDERSCORES

refer to https://solidity.readthedocs.io/en/v0.5.17/style-guide.html#naming-conventions

Examples:

Constants like :

- contract `CTokenStorage`: `borrowRateMaxMantissa`, `reserveFactorMaxMantissa`,
- contract `CTokenInterface`: `isCToken`,
- contract `Comptroller`: `compClaimThreshold`, `compInitialIndex`, `closeFactorMinMantissa`, `closeFactorMaxMantissa`, `collateralFactorMaxMantissa`, `liquidationIncentiveMinMantissa`, `liquidationIncentiveMaxMantissa`,
- contract `ComptrollerInterface`: `isComptroller`,
- contract `Exponential` `expScale`, `doubleScale`, `halfExpScale`, `mantissaOne`,
- contract `InterestRateModel`: `isInterestRateModel`,
- contract `PriceOracle`: `isPriceOracle`,

Recommendation

The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

Alleviation

The team acknowledged this issue and they stated:

"The impact of this problem is minimal. Given the contract has been deployed, it will not be modified for now.“

CJC-01 | Misuse Of A Boolean Constant

Category	Severity	Location	Status
Coding Style	Informational	Comptroller.sol	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.

For example:

```
        if (false) {
            maxAssets = maxAssets;
        }
```

Recommendation

We recommend removing the ineffectual code.

Alleviation

The team acknowledged this issue and they stated:

"The impact of this problem is minimal. Given the contract has been deployed, it will not be modified for now.“

CJC-02 | Return Value Not Stored

Category	Severity	Location	Status
Gas Optimization	Informational	Comptroller.sol: 972	Acknowledged

Description

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

Examples:

```
function _supportMarket(CToken cToken) external returns (uint) {
  ...
  cToken.isCToken();
  ...
}
```

Recommendation

We recommend adding “require” statement for isCToken:

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

Alleviation

The team acknowledged this issue and they stated:

"The impact of this problem is minimal. Given the contract has been deployed, it will not be modified for now.“

CJC-03 | Boolean Equality

Category	Severity	Location	Status
Gas Optimization	● Informational	Comptroller.sol	ⓘ Acknowledged

Description

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

For example:

```
        if (marketToJoin.accountMembership[borrower] == true) {
            // already joined
            return Error.NO_ERROR;
        }
```

Recommendation

We recommend changing it as following:

```
        if (marketToJoin.accountMembership[borrower]) {
            ...
        }
```

Alleviation

The team acknowledged this issue and they stated:

"The impact of this problem is minimal. Given the contract has been deployed, it will not be modified for now.“

CTJ-01 | Checks-Effects-Interactions Pattern Violations

Category	Severity	Location	Status
Logical Issue	Major	CToken.sol	Resolved

Description

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

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

```
800         doTransferOut(borrower, borrowAmount);
801
802         /* We write the previously calculated values into storage */
803         accountBorrows[borrower].principal = vars.accountBorrowsNew;
804         accountBorrows[borrower].interestIndex = borrowIndex;
805         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 they added a new file `CTokenERC777.sol` to be used for the ERC777 based contracts in the commit `4d3eed6650311c1dac301d6e35b52670569195c9`.

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

Category	Severity	Location	Status
Logical Issue	Major	CToken.sol: 344	Partially Resolved

Description

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

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

```
344     function exchangeRateStoredInternal() internal view returns
(MathError, uint) {
345         uint _totalSupply = totalSupply;
346         if (_totalSupply == 0) {
347             /*
348              * If there are no tokens minted:
349              *  exchangeRate = initialExchangeRate
350              */
351             return (MathError.NO_ERROR, initialExchangeRateMantissa);
352         } else {
353             /*
354              * Otherwise:
355              *  exchangeRate = (totalCash + totalBorrows -
totalReserves) / totalSupply
356              */
357             uint totalCash = getCashPrior();
358             uint cashPlusBorrowsMinusReserves;
359             Exp memory exchangeRate;
360             MathError mathErr;
361
362             (mathErr, cashPlusBorrowsMinusReserves) =
addThenSubUInt(totalCash, totalBorrows, totalReserves);
363             if (mathErr != MathError.NO_ERROR) {
364                 return (mathErr, 0);
365             }
366
367             (mathErr, exchangeRate) =
getExp(cashPlusBorrowsMinusReserves, _totalSupply);
```

```
368             if (mathErr != MathError.NO_ERROR) {
369                 return (mathErr, 0);
370             }
371
372             return (MathError.NO_ERROR, exchangeRate.mantissa);
373         }
374     }
```

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 stated:

"We will lock up a little bit of the underlying assets in each market to avoid the scenarios mentioned."

Although Compound has the same code issue, we recommend the team take care of it to prevent risks.

These market contracts are deployed at the following addresses:

https://tronscan.io/#/contract/TE2RzoSV3wFK99w6J9UnnZ4vLfXYoxvRwP

https://tronscan.io/#/contract/TXJgMdjVX5dKiQaUi9QobwNxtSQaFqccvd

https://tronscan.io/#/contract/TGBr8uh9jBVHJhhkwSJvQN2ZAKzVkxDmno

https://tronscan.io/#/contract/TLeEu311Cbw63BcmMHDgDLu7fnk9fqGcqT

https://tronscan.io/#/contract/TWQhCXaWz4eHK4Kd1ErSDHjMFPoPc9czts

https://tronscan.io/#/contract/TUY54PVeH6WCcYCd6ZXXoBDsHytN9V5PXt

https://tronscan.io/#/contract/TNSBA6KvSvMoTqQcEgpVK7VhHT3z7wifxy

GAG-01 | Centralization Related Risks

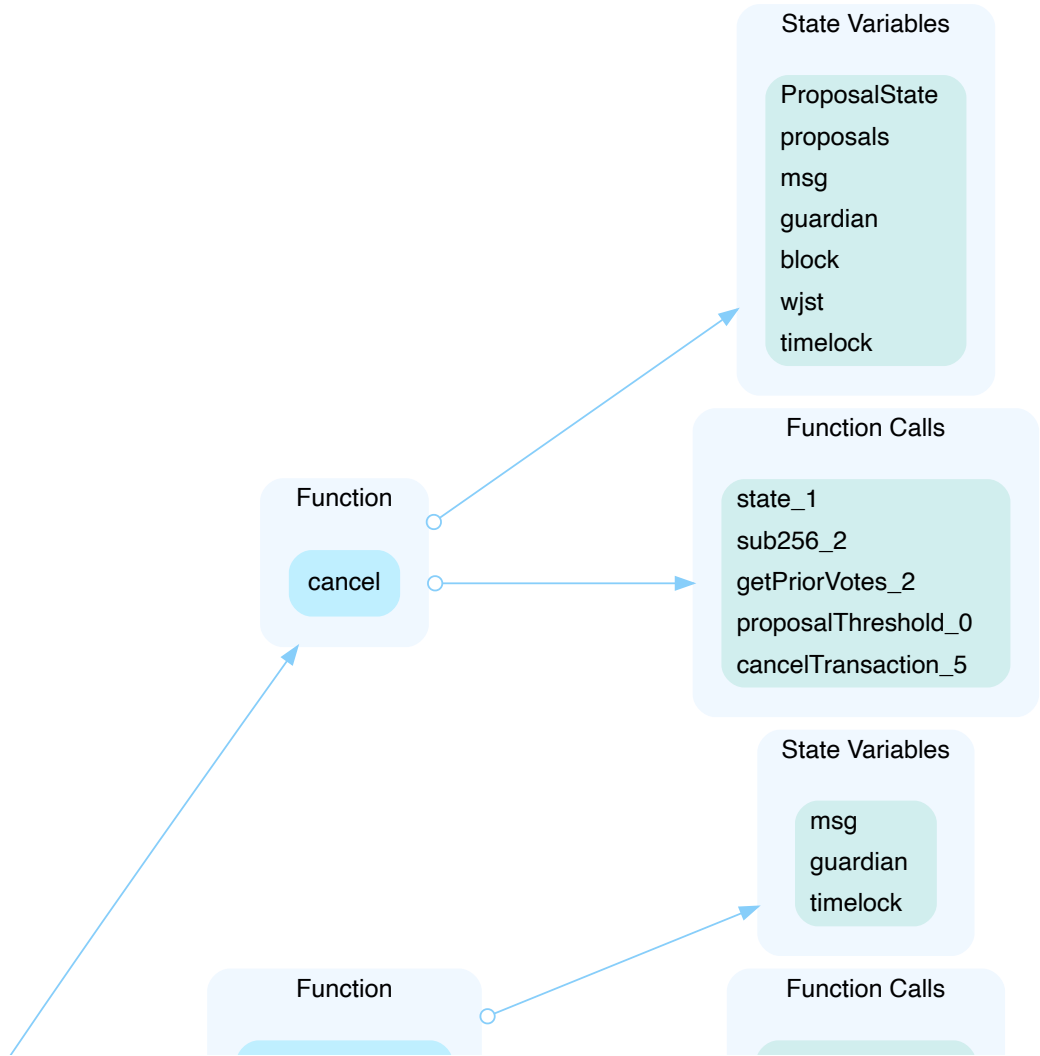
Category	Severity	Location	Status
Centralization / Privilege	**Major**	Governance/GovernorAlpha.sol: 213~226, 295~298, 300~303, 305~308, 310~313	Resolved

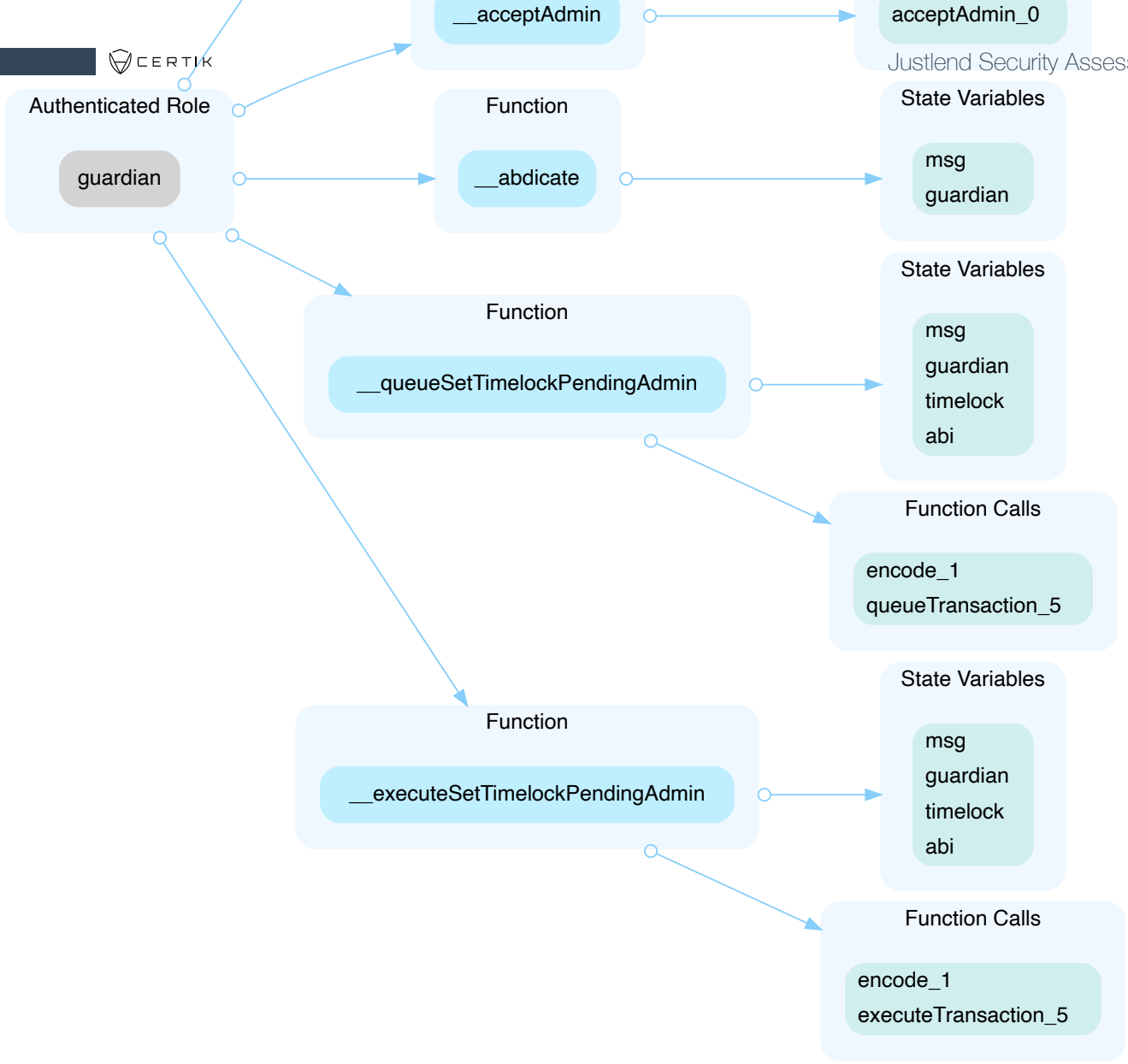
Description

In the contract `GovernorAlpha` the role `guardian` has authority over the functions shown in the diagram below.

- `cancel()`, to cancel the proposal.
- `__acceptAdmin()`, to accept admin of the `Timelock` contract.
- `__abdicate()`, to renounce `guardian`.
- `__queueSetTimelockPendingAdmin()`, to queue the transaction for `Timelock.setPendingAdmin()`.
- `__executeSetTimelockPendingAdmin()`, to execute the transaction for `Timelock.setPendingAdmin()`.

Any compromise to the `guardian` 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 heeded our advice and renounced the role `guardian` to zero address to resolve this issue.

POP-01 | Centralization Related Risks

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

Description

In the contracts `PriceOracle`, the role `anchorAdmin` has the authority over the following function:

- `_setPendingAnchor()`: set the anchor price for an asset,
- `_setPaused()`: pause or resume the market,

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

In the contracts `PriceOracle`, the role `poster` has the authority over the following function:

- `setPrice()`: set price for an asset,
- `setPrices()`: set prices for a variable number of assets

Any compromise to the `poster` 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., 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 stated:

"In Compound's current PriceOracle, only authorized addresses can submit prices. In JustLend, they use the poster role to submit prices.

In the same way, Compound currently retains the role of `anchorAdmin` to intervene in feeding prices in the event of significant price deviations. They also retain the role of `anchorAdmin` for the same purpose in JustLend, and they will upgrade this role to multi-signature in the future."

POP-02 | Logical Issue Of `setPriceInternal()`

Category	Severity	Location	Status
Control Flow	Minor	PriceOracle/PriceOracle.sol: 746	Acknowledged

Description

In the function `setPriceInternal()`, it will not save the posted price when it exceeds the maximum swing(currently: 10%). So it cannot guarantee that the posted price is between the `price - max swing` and `price + max swing`, and the posted price could be changed to the max price when it exceeds the max swing. The real asset price may exceed the maximum swing.

```
 760 if (localVars.pendingAnchorMantissa != 0) {
 761   // let's explicitly set to 0 rather than relying on default of
declaration
 762   localVars.anchorPeriod = 0;
 763   localVars.anchorPrice = Exp({mantissa :
localVars.pendingAnchorMantissa});
 764
 765   // Verify movement is within max swing of pending anchor (currently:
10%)
 766   (err, localVars.swing) = calculateSwing(localVars.anchorPrice,
localVars.price);
 767   if (err != Error.NO_ERROR) {
 768       return failOracleWithDetails(asset,
OracleError.FAILED_TO_SET_PRICE,
OracleFailureInfo.SET_PRICE_CALCULATE_SWING, uint(err));
 769   }
 770
 771   // Fail when swing > maxSwing
 772   if (greaterThanExp(localVars.swing, maxSwing)) {
 773       return failOracleWithDetails(asset,
OracleError.FAILED_TO_SET_PRICE,
OracleFailureInfo.SET_PRICE_MAX_SWING_CHECK, localVars.swing.mantissa);
 774   }
 775 } else {
 776   localVars.anchorPeriod = anchors[asset].period;
 777   localVars.anchorPrice = Exp({mantissa :
anchors[asset].priceMantissa});
 778
 779   if (localVars.anchorPeriod != 0) {
```

```
780         (err, localVars.priceCapped, localVars.price) =
capToMax(localVars.anchorPrice, localVars.price);
781         if (err != Error.NO_ERROR) {
782             return failOracleWithDetails(asset,
OracleError.FAILED_TO_SET_PRICE, OracleFailureInfo.SET_PRICE_CAP_TO_MAX,
uint(err));
783         }
784         if (localVars.priceCapped) {
785             // save for use in log
786             localVars.cappingAnchorPriceMantissa =
localVars.anchorPrice.mantissa;
787         }
788     } else {
789         // Setting first price. Accept as is (already assigned above
from requestedPriceMantissa) and use as anchor
790         localVars.anchorPrice = Exp({mantissa :
requestedPriceMantissa});
791     }
792 }
```

Recommendation

We would like the team provides more details for the control flow of price oracle.

Alleviation

The team acknowledged this issue and they stated:

"The impact of this problem is minimal. Given the contract has been deployed, it will not be modified for now.

In addition, the 30-minute fluctuation of only 10% up or down is more beneficial to the safety of the user's assets. This is because it keeps the price relatively stable and avoids unnecessary blowouts when the real price fluctuates sharply for a short period of time."

WJS-01 | Centralization Related Risks

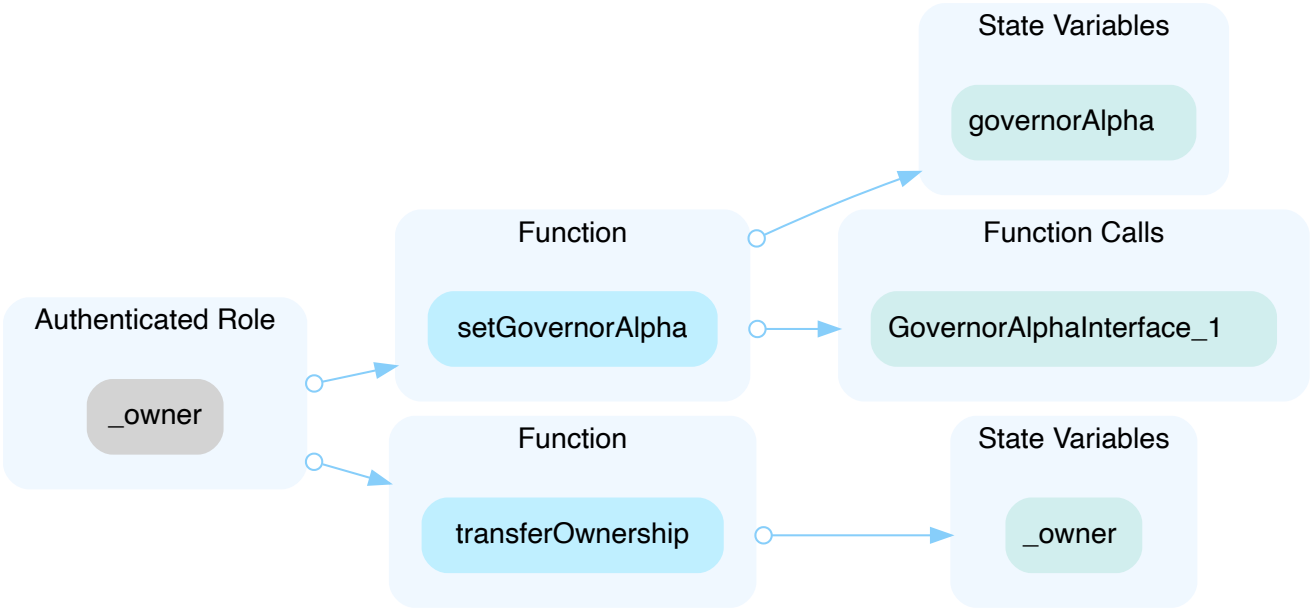
Category	Severity	Location	Status
Centralization / Privilege	**Major**	Governance/WJST.sol: 194~196, 198~202	Mitigated

Description

In the contract `WJST` the role `_owner` has authority over the functions shown in the diagram below.

- `setGovernorAlpha()`, to set the `GovernorAlphaInterface` contract.
- `transferOwnership()`, to transfer ownership to another account.

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 the issue and adopted the Timelock solution to delay-sensitive operations at the current stage. The `WJST` contract has transferred the ownership to a Timelock contract with a minimal 48 hours delay. And the Timelock contract has transferred the ownership to a governance contract to increase transparency and user involvement.

The contract `WJST` is deployed at `https://tronscan.org/#/contract/TCczUFrX1u4v1mzjBVXsiVyehj1vCaNxDt`.

The admin of contract `WJST` is a Timelock contract, which is deployed at `https://tronscan.org/#/contract/TRWNvb15NmfNKNLhQpxefFz7cNjrYjEw7x`.

The admin of contract `Timelock` is a governance contract, which is deployed at `https://tronscan.org/#/contract/TH1SVVVU9NF1ans3CRBCJ5kW2yvn4sHP9b`.

WJS-02 | Vote For Multiple Active Proposals

Category	Severity	Location	Status
Control Flow	Informational	Governance/WJST.sol: 109	Acknowledged

Description

The users will deposit `JST` token to mint `WJST`. They need to lock `WJST` token to the `WJST` contract to vote for a proposal. If there are multiple active proposals, the user's votes are locked in the voted proposal, but cannot vote for the others.

Recommendation

We recommend stating for this.

Alleviation

The team acknowledged this issue and they stated:

"The impact of this problem is minimal. Given the contract has been deployed, it will not be modified for now.“

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.

Control Flow

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

Volatile Code

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

Data Flow

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

Language Specific

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

Coding Style

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

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.