Smart Contract Audit

coinspect

Tropykus

Tropykus

Smart Contract Audit

V211208

Prepared for Tropykus • November 2021

1. Executive Summary

In **November 2021, Tropykus** engaged Coinspect to perform a source code review of their Protocol lending platform.. The objective of the project was to evaluate the security of the smart contracts.

Tropykus is a fork of a previously audited project, rLending, which is a fork of the Compound project. A previous version of Tropykus was already audited by Coinspect. This incremental audit focuses on the changes introduced since Coinspect's last review.

The following issues were identified during the assessment:

High Risk	Medium Risk	Low Risk
1	2	1
Fixed: 1	Fixed: 1	Fixed: 1

The high risk issue `TRO-4` warns about not verifying the value sent in a transaction. The medium risk issue `TRO-5` refers to wrong assumptions made about the RSK blockchain. The medium risk issue `TRO-6` and the low risk `TRO-7` caution about bad error handling.

2. Assessment and Scope

The audit started on November 15, 2021 and was conducted on the tag `v0.2.4` of the git repository at https://github.com/Tropykus/protocol as of commit `6f61541e7fbfd68c85e6e7c66d9e8f167a536a44` of **November 14, 2021**.

The aggregated file changes are shown with their respective sha256 hash below

```
9b4466b38cadb046fdea51416ad38c58417fb9826d24215d89d29048f9a1e6b8  contracts/CCompLikeDelegate.sol
527e4240980a202d28e4fe10f107e0c7852b296fda8a7bacdcfe78519aaf18f1  contracts/CErc20.sol
bf3b9fd86d7d3ba1340ca7a9e4e76d5210392b1c83aef76a606a4b2ffd97727f  contracts/CErc20Delegate.sol
c4488c256dfb38e071dffd7642d03beb491349acfb736f02308d85266add76f7  contracts/CRBTC.sol
857fb7bc87ec044d6fc6bff354f9a65526972827b313da23f07c50f2c0775df7  contracts/CRBTCCompanion.sol
da01bb6d70c4e5507023a3281820d87bdc8d9985a42ce55f32d85e7d82f46e61
contracts/CRBTCCompanionInterface.sol
e054c7f837a8707dae835c9e78098dcf0f23baa701a66521782382b844aa23d4  contracts/CRDOC.sol
b5b608e8b9c6c63494aafda3ce4e632d15dc140db6c753fa5ae4dd50aa71562c  contracts/CToken.sol
8f20e6c304df72cb63c15be5394677f7b351e4169436cd73e937dba8e7975256  contracts/Comptroller.sol
0e667da76fe3a26aa03fab5a97c31d66527fbcaebc834f83c026acbf4b584716  contracts/ComptrollerG1.sol
c52d57b7cafcd7ab45ba7ebaa35270c5ec0a61744afce1c12d5ff67a2e58c6c1  contracts/ComptrollerG2.sol
51bfbb8f6a6603a7cee77da13babc31c70b9236671f9ed374896a5ac10ac5d6c  contracts/ComptrollerG3.sol
60fd605e7f3ef05834743181cf63502132fa3e1874316e7530c8249d060c1ec8  contracts/ComptrollerG4.sol
6c26f79ab65cb4fe97be5502cc920931ffe2b542a53e22be8b34f3f047566213  contracts/ComptrollerG5.sol
93a40f2b4784a287b76066091aef9f6edcb93f6f2ea567407ab6eeb1cd1fde5d  contracts/ComptrollerG6.sol
b49f47b93ddbfb965a741de9a49a8d9ead695af11f0ee51360c2b9e56a6b1cda  contracts/Governance/TROP.sol
```

On December 6, 2021 an update with fixes was submitted for review. The corresponding sha256 hashes of the updated files are shown below.

```
0cebef1032e555ff9197c403b619347da7eecf71be878039df594f3a00ad15d5  CRBTC.sol
d738e9626f79940156cc442aa7a819472da74948ad8b2f7f6958a44808db8216  CToken.sol
2419423352373dfef33a5b2870bfcc2ef4fd9cdacebb2d4ba9810045645a4dbd  BaseJumpRateModelV2.sol
502240407f21107952627196f70a985d540a027f677699f2c411729c91959390  HurricaneInterestRateModel.sol
ec62aa919d359d30158a51912b31948cbe5d05d05a3bdce8b1738b6bff725d3f  InterestRateModel.sol
aa9f60fd3cad493181db3235e34cf7729cacd4e8fb378bbfae51dc0763e9ba8d  JumpRateModel.sol
14abaa64a2a9a94b401495c743b288137f55853d6534a4dc55ec1b68348b8ca2  JumpRateModelV2.sol
b416197ac83f95a40059569faacb844400d13be644acfc09fa333bd0abf37e4e  WhitePaperInterestRateModel.sol
```

The smart contracts are compiled with an outdated Solidity compiler version: 0.5.16. Newer compiler versions include several gas usage optimizations, compile time warnings improvements and bug fixes. For example, a recently published security issue was fixed in Solidity compiler 0.8.4 and affects all previous versions (see Solidity ABI Decoder Bug For Multi-Dimensional Memory Arrays for more information).

Tropykus is based on rLending and Compound protocol and will be deployed the RSK blockchain. Tropykus implements a new interest model called Hurricane

Interest Rate Model with guarantees of a Minimum Interest Rate provided by a subsidized fund.

The main documentation used is the whitepaper provided by the Tropykus team. The motivation and details of the Hurricane Interest Rate Model can be found there.

The largest set of changes introduced by Tropykus are for supporting multiple interest rates for the `CToken` contract and the implementation of the Hurricane Interest Rate Model.

The new version includes:

- Limits on `rBTC` lending
- Limits on `RDOC` borrowing
- Reduced smart contract size
- Changed error strings for error codes
- New logic for redeeming and accruing interest when using the Hurricane rate model
- Bug fixes

A fixed block time is assumed by the code, but in the RSK network the average block time is not as stable as in Ethereum and it could change, leading to `TRO-5`, an issue where the interest rate can be miscalculated.

The changes that added the `repayBorrowAll` function also missed an important validation about the value sent on the `CRBTC` contract, allowing users to steal funds from the contract (see `TRO-4`).

It is important to note that limiting minting per address is ineffective as the same user can split the minting desired in any number of addresses.

3. Summary of Findings

Id	Title	Total Risk	Fixed
TRO-4	RBTC can be stolen	High	✔
TRO-5	Blocks per year might lead to wrong interest model	Medium	✓
TRO-6	Unhandled math errors	Low	✔
TRO-7	Deposit subsidy might be ignored by CRBTC	Medium	✔

4. Detailed Findings

TRO-4 RBTC can be stolen

Total Risk	Impact	Location
High	High	`contracts/CRBTC.sol`
Fixed	Likelihood	
✔	High	

Description

Attackers can steal `rBTC` by calling the repaying borrow function without actually returning funds.

When calling the `repayBorrowAll` function in the `CRBTC` contract, the contract never checks that the correct amount was sent in the transaction.

The `repayBorrowAll` function calls the internal `repayBorrowFresh` method with the `repayAmount` equals to `uint256(-1)`. The function calls an inherited method to check the transferred amount,

[contracts/CToken.sol]

```
1594  if (repayAmount == uint256(-1)) {
1595      vars.repayAmount = vars.accountBorrows;
1596      vars.actualRepayAmount = doTransferIn(
1597              payer,
1598              vars.repayAmount,
1599              true
1600          );
1601  }
```

But in the implementation the value is not checked.

[contracts/CRBTC.sol]

```
192  function doTransferIn(
193              address from,
194              uint256 amount,
195              bool isMax
196          ) internal returns (uint256) {
197      isMax;
```

```
198       // Sanity checks
199       require(msg.sender == from, "sender mismatch");
200       return amount;
201   }
```

The `msg.value` is never validated through all the `repayBorrowAll` calls.

Recommendation

Check that the funds were sent within the transaction.

Status

Fixed in commit `f00b7d30713f90af9fd7f9872f7a4c7674d579de`

TRO-5	Blocks per year might lead to wrong interest model		
Total Risk Medium	Impact Medium	Location `contracts/InterestRateModel.sol`	
Fixed ✓	Likelihood High		

Description

A wrong interest rate might be accrued by an incorrect assumption about the number of blocks per year.

The difficulty adjustment algorithm used by the RSK network does not guarantee an average time between mainchain blocks, it targets a given density of blocks including trunk and ommen blocks.

In RSK, most miners are configured to minimize mining pool bandwidth and create a high number of ommers. This is permitted by design. They can also be configured to minimize the number of ommers, and consume more bandwidth. RSK targets approximately a density of 2 blocks every 33 seconds, and currently one block is an ommer, and the other is part of the trunk.

If miners decide to update their configurations to minimize ommer blocks the average block time may go down to 16.5, making the number of the block unreliable as a time measure.

Recommendation

Use the block time value instead of the block number for calculating the accrued interest.

Status

Partially fixed in commits `3785220a20abb08f0bdcbb93891e8678066456aa` and `cb40e0a3413b043960e8b1a8ac8acfade9aebefe.` Instead of using the block time Tropykus team decided to make the `blocksPerYear` variable updatable. This is not

a solution per se, but allows the admins of the network to solve the issue through monitoring and updating this value.

Additionally `setBlocksPerYear` does not emit any event. This makes it difficult for users to track changes on the value that might be significant for them.

TRO-6 Unhandled math errors

Total Risk	Impact	Location
Low	Low	contracts/CToken.sol
Fixed	**Likelihood**	
✔	Medium	

Description

The `CarefulMath` and `Exponential` libraries provide methods for safely performing math operations similar to OpenZeppelin `SafeMath` library but with different error handling. Where the `SafeMath` lib reverts, the `CarefulMath` lib returns an error.

In the `tropykusInterestAccrued` function all errors returned by `CarefulMath` and `Exponential` are ignored.

```
[contracts/CToken.sol]
508  function tropykusInterestAccrued(address account)
509          public
510          view
511          returns (
512              MathError,
513              uint256,
514              uint256,
515              uint256,
516              uint256
517          )
518  {
519      SupplySnapshot storage supplySnapshot = accountTokens[account];
520      uint256 promisedSupplyRate = supplySnapshot.promisedSupplyRate;
521      Exp memory expectedSupplyRatePerBlock = Exp({
522              mantissa: promisedSupplyRate
523      });
524      (, uint256 delta) = subUInt(
525              accrualBlockNumber,
526              supplySnapshot.suppliedAt
527      );
528      (, Exp memory expectedSupplyRatePerBlockWithDelta) = mulScalar(
529              expectedSupplyRatePerBlock,
530              delta
531      );
532      (, Exp memory interestFactor) = addExp(
533      Exp({mantissa: 1e18}),
534              expectedSupplyRatePerBlockWithDelta
535      );
536      uint256 currentUnderlying = supplySnapshot.underlyingAmount;
537      Exp memory redeemerUnderlying = Exp({mantissa: currentUnderlying});
538      (, Exp memory realAmount) = mulExp(interestFactor, redeemerUnderlying);
```

```
539      (, uint256 interestEarned) = subUInt(
540              realAmount.mantissa,
541              currentUnderlying
542      );
543      (, Exp memory exchangeRate) = getExp(
544              realAmount.mantissa,
545              supplySnapshot.tokens
546      );
547      return (
548              MathError.NO_ERROR,
549              interestFactor.mantissa,
550              interestEarned,
551              exchangeRate.mantissa,
552              realAmount.mantissa
553      );
554  }
```

There is a similar issue in the `tropykusExchangeRateStoredInternal` function where errors are ignored. The caller even assumes that the only possible error is when there is no supply as shown below.

```
456  (error, exchangeRate) = tropykusExchangeRateStoredInternal(
457          msg.sender
458  );
459  if (error == MathError.NO_ERROR) {
460      return (MathError.NO_ERROR, exchangeRate);
461  } else {
462      return (MathError.NO_ERROR, initialExchangeRateMantissa);
463  }
```

Coinspect did not confirm the exploitability of these issues, but improper error handling increases security risks.

Recommendation

Do not ignore errors that could lead to security issues.

Status

Fixed at commit `c892950eeb78045a08c429ff2f2cf7f73bb0cb12`.

TRO-7 Deposit subsidy might be ignored by CRBTC

Total Risk	Impact	Location
Medium	High	`contracts/CRBTC.sol`
Fixed	Likelihood	
✔	Low	

Description

If a user deposits a subsidy in the `CRBTC` contract it might be ignored.

The `addSubsidy` function calls the `addSubsidyInternal` which can return an error when accruing interest, but the caller does not revert appropriately.

[contracts/CRBTC.sol]

```
232  function addSubsidy() external payable {
233       _addSubsidyInternal(msg.value);
234  }
```

[contract/CToken.sol]

```
2215  function _addSubsidyInternal(uint256 addAmount)
2216          internal
2217          nonReentrant
2218          returns (uint256)
2219  {
2220      uint256 error = accrueInterest();
2221      if (error != uint256(Error.NO_ERROR)) {
2222          // accrueInterest emits logs on errors...
2223          return fail(Error(error), FailureInfo.ADD_SUBSIDY_FUND_FAILED);
2224      }
2225
2226      (error, ) = _addSubsidyFresh(addAmount);
2227      return error;
2228  }
```

In the case that an error is returned in line 2223, the money transferred within the transaction is kept in the contract, but without the side effects of `_addSubsidyFresh`.

Recommendation

Revert on error in `addSubsidy`.

Status

Fixed at commit `f2b2e5b773b29c15af6b76e432b4bc5f32541873`.

5. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.