Smart Contract Audit

coinspect

Tropykus

Tropykus

Smart Contract Audit

Prepared for Tropykus • July 2021

v211407

1. Executive Summary

In **June 2021**, Tropykus engaged Coinspect to perform a source code review of the Tropykus Protocol lending platform.

Tropykus is a fork of a previously audited project, RLending, which is a fork of the Compound project. This incremental audit focused on the changes introduced since Coinspect's last review.

Coinspect found the modifications performed to be consistent and to not affect the original project's threat model and security assumptions. The new code additions are clearly documented.

Coinspect did not find any vulnerabilities that could allow adversaries to steal user funds.

However, as Coinspect previously reported during the RLending project audit, the current oracle infrastructure available on the RSK network represents a potential risk and constitutes a single point of failure.

2. Assessment and Scope

The audit started on June 30, 2021 and was conducted on the `master` branch of the git repository at https://github.com/Tropykus/protocol as of commit `3b2ea8bf1364b6b7af2e114ec2c550d1c9cbebfb` of **June 30, 2021.**

The Tropykus Finance Protocol whitepaper (June 18, 2021 version) was utilized as a reference during the engagement.

The smart contracts are compiled with an outdated Solidity compiler version: 0.5.16.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 Protocol will be bootstrapped by a set of “protocol sponsors” who will act as the initial administrators with the capacity to configure all contracts in the system and update all parameters. After the system matures, governance will migrate to a more decentralized model involving the protocol users.

The most important modification introduced by Tropycus is the **Hurricane interest rate model** which guarantees a base return rate for lenders. It is worth noting that Tropykus markets can use either the new Hurricane interest model or the legacy Compound interest rate model. A **subsidy fund** was added with the objective of subsidizing the promised yields to the market suppliers. The smart contracts allow anybody to add funds to any market’s subsidy fund and according to the whitepaper this funds will be initially provided by the protocol sponsors. However, the origin of funds, minimum values, and maintenance schedule are not programmed in the smart contracts in scope for this audit. The whitepaper does not make clear which will be the criteria that is going to be used for this purpose. It is worth observing that **if the subsidy funds are not enough, the protocol might fail to meet the promised rate** to suppliers.

The protocol exposed attack surface suffered minor modification. The most important modifications to the smart contracts were concentrated in the following files:

- `InterestRateModel.sol`
- `CErc20.sol`
- `CRBTC.sol`
- `CRBTC.sol`
- `CToken.sol`
- `CTokenInterfaces.sol`
- `HurricaneInterestRateModel.sol`
- `SignedSafeMath.sol`

A detailed list of the changes performed to each file is provided in the Tropykus whitepaper.

The `CToken` contract was the most heavily modified source file. Several Tropykus specific scenarios are handled after checking if the Tropykus interest model is being utilized by calling the `isTropykusInterestRateModel` function. The new `tropykusExchangeRateStoredInternal` function was added to calculate the promised supply rate. The reserves capitalization was incorporated into the `accrueInterest` function, by calling `newReserves` and checking if the utilization rate is below the optimal rate. Changes to the supply business logic were performed to the `mintFresh` and `redeemFresh` functions, which now utilize the

`subsidyFund` when appropriate, according to the information stored in `supplySnapshot`. The new `_addSubsidyInternal` function allows adding funds to the subsidy fund and can be called by anybody.
The CRBTC is a CToken implementation that wraps RBTC.

The `HurricateInterestModel` contract implements `getSupplyRate` and `getBorrowRate` functions according to the formulas in the whitepaper. The legacy functions were refactored to the `InterestRateModel` contract.

The `TROP` contract implements the project's governance token, which allows delegation for voting purposes. The contract supports EIP-712 and `delegateBySig` functionality.

Regarding the price oracles utilized by Tropykus, there have been no updates to the code handling the price data feeds, and as Coinspect previously reported during the RLending project audit, the current available infrastructure represents a potential risk and constitutes a single point of failure.
A document acknowledging this risk and presenting users a roadmap can be found in Tropykus code repository:
https://github.com/Tropykus/protocol/blob/main/audit/oracles.md.

Neither Compound Protocol's security nor the Money on Chain oracle infrastructure were evaluated during this audit.

It is recommended to **follow-up with all changes to the Compound project** in order to quickly react if a vulnerability is fixed as it is very probable that it also affects Tropykus.

3. Summary of Findings

ID	Description	Risk	Fixed
TRO-001	Outdated Solidity compiler version	Info	✘
TRO-002	ERC20 approve front running	Info	✘
TRO-003	Hardcoded COMP token address	Info	✘

4. Detailed Findings

TRO-001	Outdated Solidity compiler version	
Total Risk **Info**	Impact	Location *
Fixed ✘	Likelihood	

Description

Using outdated Solidity compiler versions might miss fixes and ignore optimizations and warnings introduced in newer versions which could help flag security vulnerabilities.

The contracts reviewed during the assessment are specified to be compiled with Solidity version 0.5.16 at least:

```
pragma solidity ^0.5.16;
```

The latest Solidity version available is 0.8.6. Version 0.5.16 is outdated, and does not contain all the checks included in the latest versions. The newer versions include changes to the language that render it safer, preventing some mistakes that could lead to security-relevant bugs. Also, bug fixes in Solidity are not backported, so it is always recommended to upgrade all code to be compatible with Solidity v.0.8.6.

Recommendation

Upgrade the contracts to the newest version of Solidity if possible.

TRO-002 ERC20 approve front running

Total Risk	Impact	Location
Info		`CToken.sol` `ERC20.sol`
Fixed	Likelihood	
✘		

Description

An attacker could leverage a well known issue in the ERC20 standard to spend more than their intended allowance when this value is updated.

The `CToken` and `StandardToken` contracts suffers from a well known ERC20 standard security vulnerability that takes place when the token transfer allowance is modified: an attacker can front run the approve transaction to transfer the original allowed amount of tokens (N) before the allowance is changed, and then, after the approve transaction takes place, the attacker can again transfer more tokens (M), obtaining as a result more tokens than the toker owner intended (N+M instead of M).

Note this issue is correctly documented in the source code:

```
    * @dev This will overwrite the approval amount for `spender`
    *  and is subject to issues noted [here](https://eips.ethereum.org/EIPS/eip-20#approve)
```

Recommendation

Add the functions `increaseApproval` and `decreaseApproval` to the affected contracts, using as a template the implementations in the OpenZeppelin library.

https://github.com/ethereum/EIPs/issues/20#issue comment-263524729
https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/StandardToken.sol#L70

TRO-003 Hardcoded COMP token address

Total Risk	Impact	Location
Info		`ComptrollerG6.sol`
Fixed	Likelihood	
✗		

Description

The address of the COMP token in Ethereum mainnet is hardcoded in the `ComptrollerG6` contract:

```
/**
 * @notice Return the address of the COMP token
 * @return The address of COMP
 */
function getCompAddress() public view returns (address) {
    return 0xc00e94Cb662C3520282E6f5717214004A7f26888;
}
```

This function is used in `transferComp` and `grantCompInternal` functions, for example:

```
function transferComp(address user, uint userAccrued, uint threshold)
         internal returns (uint) {
    if (userAccrued >= threshold && userAccrued > 0) {
        TROP comp = TROP(getCompAddress());
```

Recommendation

Define a constant to replace the hardcoded address.

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.