Compound: Tether Integration Audit

OPENZEPPELIN SECURITY | APRIL 26, 2020

Security Audits

After an initial phase, the Compound team engaged us to finish auditing their integration of Tether's USDT token into the `PriceOracleProxy` contract, together with minor modifications to the already audited codebase, such as gas optimizations and updates to implement Solidity 0.6.0 style inheritance. The audited commit is `65e870f7f45dca0a7a3a70209da9c5aec9d27c13` of Compound's private repository.

Security assumptions

Collateral tokens, transfer fees, and the liquidation incentive

Stablecoins make a natural choice for the tokens used as collateral in compound. One of the most popular stablecoins, USDT, is capable of extracting a fee from the recipient of a transfer or transferFrom (see line 131 of the USDT contract code). While USDT is not doing this at the time of writing, it will remain capable of doing so unless the owner of the USDT contract provably burns their ownership.

An earlier refactor of Compound's `doTransferIn` function was done to correctly handle the case where the underlying token extracts a fee from the recipient upon a `transfer` or `transferFrom`, so Compound's internal accounting will remain correct even if such a fee is levied.

profitable. If the transfer fee were greater than Compound's liquidation incentive, then liquidation would not be profitable and the liquidation mechanism could fail.

Currently, Compound does not intend to allow USDT to be used as collateral. Even if they did, USDT's maximum transfer fee is 50 basis points (0.5%), while Compound's liquidation incentive is currently 8%. So this would not threaten the incentive compatibility of the liquidation mechanism.

However, this dynamic should be kept in mind when adding new collateral tokens to Compound and when adjusting Compound's liquidation incentive. In this report, we will assume that the liquidation incentive will always be greater than any fee that is levied by a token that may be used as collateral.

USDC and USDT price stability

The new code fixes the reported prices of USDT and USDC to $1 USD. That is, rather than having an "active", trusted third-party price oracle that detects the real market prices of USDT and USDC and reports them to Compound, the contracts will instead always assume that the market prices of those assets are $1 USD.

Using an "active" trusted third-party price oracle carries its own risks, which we've discussed in previous audits. Assuming that the USDT and USDC stablecoins remain perfectly pegged at $1 USD comes with a different set of risks.

For example, if the real market price of one of these stablecoins exceeds $1 (which may happen, for example, if a large exchange freezes withdrawals for all asset other than the stablecoin, thus causing high demand for the stablecoin on the exchange), then some Compound borrows may become undercollateralized without Compound's contracts detecting it, and so liquidation of those borrows would not be possible. Similarly, if the real market price of a stablecoin (that is being used as collateral) drops sufficiently below $1 USD, then some borrows may become undercollateralized without Compound's contracts detecting it, and so liquidation of those borrows would not be possible. In an extreme case where the real market price of one of these stablecoins drops below its collateral factor, some borrows may become entirely underwater without Compound's contracts detecting it, potentially threatening Compound's solvency.

trusted third-party oracle. This is the motivation behind using a fixed-price for these assets rather than using an “active” price oracle.

This introduces a new security assumption. In particular, we assume that the real market prices of USDC and USDT will remain sufficiently close to $1 USD that the above-mentioned issues do not materialize. However, we do recommend that the Compound team actively monitor the real market prices of these assets and be ready to pause affected markets in the event that real market prices deviate significantly from $1 USD.

High level overview of the changes

The audited patch consists of minor changes to “Price Oracle” contracts `PriceOracleProxy`, `PriceOracle` and `SimplePriceOracle`. The patch readies the contracts for USDT integration, modifies the `accrueInterest` function of the `CToken` contract, and removes the `checkTransferIn` function. It also includes gas optimization changes in both the `CToken` and the `CErc20Delegator` contracts, including reducing the number of SLOADs within `CToken`, and having `CErc20Delegator` use `msg.data` directly, which eliminates unnecessary repacking. Finally, as checked in the previous audit phase, the patch eliminates the use of `localVars` structs throughout the code, removes any instances of `constant`s being declared within interfaces, bumps the Solidity version to 0.5.16 in some contracts, and switches many instances of `CarefulMath` to use `SafeMath` instead.

We did not identify any security issues regarding the changes to `accrueInterest`, the removal of `checkTransferIn`, or the various small optimizations made to the code in this commit.

For validating the gas optimizations changes, we confirmed that the `CToken` contract does maintain the same behavior after the replacement of storage variables with variables in memory, and that the functionality of `doTransferIn` does obviate the need for `checkTransferIn`. We also validated that the `CErc20Delegator` contract still correctly delegates calls to the implementation.

A detailed list with all the changes can be found in the commit message

Critical severity

None. 🙂

High severity

None. 🙂

Medium severity

[M01] Undocumented assembly blocks

The `CErc20Delegator` contract includes multiple assembly blocks.

While this does not pose a security risk per se and the code does not present vulnerabilities after thorough review, these blocks are a critical part of the system and should be better documented. Moreover, as this is a low-level language that is harder to parse by readers, consider including extensive documentation regarding the rationale behind its use, clearly explaining what every single assembly instruction does. This will make it easier for users to trust the code, for reviewers to verify it, and for developers to build on top of it or update it.

In particular, consider explaining:

- Why the arguments have to be moved from calldata.
- How `delegateAndReturn` and `delegateToViewAndReturn` provide the return value without the `return` keyword.
- That the `0x20` hardcoded value in the reverts of `delegateTo` and `delegateToViewImplementation` exists to offset a bytes array, and the reason for this offset.
- Why this offset is not used in the reverts of the `delegateToViewAndReturn` and `delegateAndReturn` functions.
- Why the return is displaced by `0x40` in the `delegateToViewAndReturn` function and why it is not needed in the `delegateAndReturn` function.

all potential use cases of these functions to ensure they behave as expected.

Low severity

[L01] Price manipulation possible in `SimplePriceOracle`

The `setUnderlyingPrice` and `setDirectPrice` functions from the `SimplePriceOracle` contract allow the caller to set asset prices in the contract storage. However, these functions do not implement any access control mechanism and have their visibility set as `public`, thus allowing anyone to execute them.

Any contract using this Oracle to determine the price of an asset could be subject to price manipulation attacks by anyone.

Although the Compound team has explained that this contract is only to be deployed on testnets, consider specifying this through comments or documentation, and consider enforcing this programmatically, perhaps by ensuring that `chainid` is not equal to 1.
For the general public and the whole DeFi space, we highlight that this contract must not be deployed into mainnet.

[L02] Lack of indexed parameters in `PricePosted` event

None of the parameters in the `PricePosted` event defined in the `SimplePriceOracle` contract are indexed.

Consider indexing the `asset` parameter to ease the task of searching and filtering for specific events.

[L03] The SAI Price is Unchangeable after SCD shutdown

In the `PriceOracleProxy` contract, `setSaiPrice` is able to be called once, upon Single Collateral Dai shutdown, to set the price of SAI within Compound. Afterwards, this price cannot be changed. It is intended that the peg be set to equal the exchange rate at which Maker will purchase SAI for ETH after SCD shutdown.

Although it is unlikely that the actual market price of SAI will deviate from the peg, there is a chance that it could. If it does, Compound will have no way to change the internal price set for SAI without updating the price via governance. If SAI's real market price increases, it may cause confusion as users will be able to borrow less than they anticipate, and be liquidated sooner than anticipated, when using SAI as collateral. If SAI's real market price drops, users holding SAI as collateral may be able to borrow more value than the SAI they hold is worth, creating insolvency risk for the protocol.

In the unlikely event of SAI's price dropping, and assuming Compound holds a total of 600,000 SAI with a SAI collateral factor of `0.75`, Compound would be at risk of losing at most $450,000 USD worth of value. Please note that this will be measured in ETH value post-SCD shutdown, so these figures may change. However for comparison, Compound currently custodies about $115M USD worth of assets, so this risk is small relative to the overall size of all Compound markets.

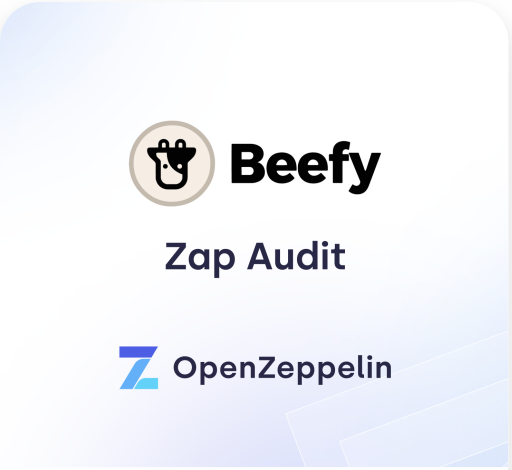
In the event that SAI's market price diverges from the peg, consider initiating a governance proposal to change the oracle to one that actively reports the real market price of SAI. Additionally, consider using the `borrowAllowed` hook to prevent users from any further borrowing until their `balanceOf` cSAI is `0`.

Notes & Additional Information

None. 🙂

Conclusion

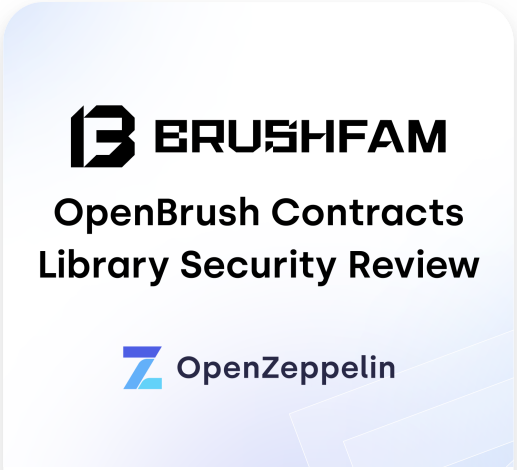
No critical or high severity issues were found. Recommendations were made to mitigate risks related to edge cases involving specific market conditions, and to improve the project's overall quality and robustness. We recommended monitoring the real market prices of USDT and USDC and pausing Compound markets if those prices deviate significantly from $1 USD. Overall we found the code to be very clean, well-organized, and easy to follow.



Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

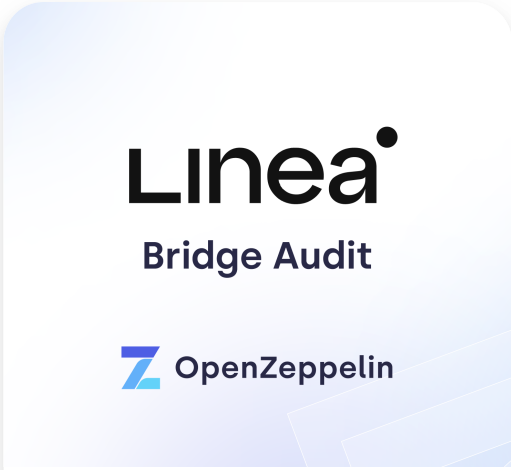
Security Audits



OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits



Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits