Compound Finance – Timelock Audit

OPENZEPPELIN SECURITY | OCTOBER 23, 2019

Security Audits

Compound Finance is a protocol, currently deployed on the Ethereum network, for automatic, permissionless loans of Ether and various ERC20 tokens. It is one of the most widely used decentralized finance systems in the ecosystem.

We previously audited a subset of Compound's contracts at commit `f385d71983ae5c5799faae9b2dfea43e5cf75262` of their public repo. Since then, the Compound team has made changes to the contract code. Those changes are reflected in commit `f244c2270f905287cb731d8fd3693ac77f8404f9`.

This audit covers only the difference between commit `f385d71983ae5c5799faae9b2dfea43e5cf75262` and commit `f244c2270f905287cb731d8fd3693ac77f8404f9`. The contracts included in the scope were: CErc20, CEther, CToken, Comptroller, ComptrollerV2Storage, ComptrollerV1Storage, ComptrollerErrorReporter, Exponential, Timelock, Unitroller.

Here we present only the new issues that we found when auditing this code upgrade. Issues found in our previous report may still apply. This audit does not cover the full contract code. It covers only the patch mentioned above.

High-level overview of the patch

This patch introduces a Timelock contract that allows its `admin` to add arbitrary function calls to a queue. The Timelock contract can only execute a function call if the function call has been in the queue for at least two days. The idea is that anytime the Timelock contract makes a function call, it must be the case that the function call was first made public by having been publicly added to the queue at least two days prior.

The intention is to have the `admin` of all CToken, Comptroller, and Unitroller contracts be an instance of the Timelock contract. This would mean that any changes made by any admin of any Compound contract would necessarily come with at least a two-day advanced warning. This makes the Compound system follow a “time-delayed, opt-out” upgrade pattern (rather than its current “instant, forced” upgrade pattern).

Time-delaying admin actions gives users a chance to exit Compound if its admins become malicious or compromised (or make a change that the users do not like). However, it also means that honest admins would be unable to lock down functionality to protect users if a critical bug was found. To address this issue, the patch introduces a new privileged role called the “pauseGuardian” in the Comptroller contract.

The pauseGuardian is *not* intended to be an instance of Timelock. It has the ability to (instantly) pause/unpause the following four functionalities: minting, borrowing, transferring CTokens, and liquidating *(Note that this has since been changed, please see “Update” below).* The ability to pause minting, borrowing, and transferring of CTokens collectively constitutes the ability to prevent users from *entering* any new positions in Compound.

Importantly, the pauseGuardian does *not* have the ability to prevent users from *exiting* their positions by calling `redeem` or `repayBorrow`. (A notable exception is when the asset underlying the CToken is another CToken with the same Comptroller. We discuss this edge case below).

This security analysis assumes the following about the Timelock, CToken, Unitroller, and Comptroller contracts:

For all CTokens, the CToken `admin` is an instance of Timelock. The CToken `comptroller` is an instance of Unitroller. The Unitroller `admin` is an instance of Timelock. The Unitroller `comptrollerImplementation` is an instance of Comptroller. The Comptroller `admin` is an instance of Timelock. The Comptroller `pauseGuardian` is *not* an instance of Timelock. The Timelock `admin` is *not* an instance of Timelock.

Here we present our findings.

Update: *The Compound team has swapped the ability to pause liquidation (directly) for the ability to pause `seize`. Since liquidating requires calling `seize`, pausing `seize` also (indirectly) pauses liquidation. Pausing `seize` rather than liquidation may help guard against a strictly larger class of potential bugs (for example, bugs that may exist in other listed CTokens), without preventing users from exiting via `redeem` or `repayBorrow`.*

Update: *The ability to unpause various functionalities has been removed from the pauseGuardian role. Instead, to unpause any paused functionality, a call must be made by the Comptroller admin. Additionally, the Comptroller admin is now able to pause all pausable functionalities, although this will incur a delay if the admin is an instance of Timelock. All pause/unpause functions can be found in Comptroller.sol between lines 998 and 1032.*

Critical

None. 🙂

High

None. 🙂

Medium

pauseGuardian. To replace the current pauseGuardian, the Comptroller admin must first call `_setPendingPauseGuardian`, passing in a new, honest pauseGuardian address. Then, the new, honest pauseGuardian must call `_acceptPauseGuardian`.

If the current, malicious pauseGuardian can call `_setPendingPauseGuardian` (passing in any address they control) *after* the Comptroller admin calls `_setPendingPauseGuardian` and *before* the honest pauseGuardian calls `_acceptPauseGuardian`, then the malicious pauseGuardian maintains control of the role. In other words, once the Comptroller admin calls `_setPendingPauseGuardian`, there exists a race between the new, honest `pendingPauseGuardian` and the existing, malicious pauseGuardian to determine who gets the pauseGuardian role moving forward.

The admin can use a contract to call both `_setPendingPauseGuardian` and `_acceptPauseGuardian` in a single transaction, but that would be non-trivial.

If a malicious/compromised pauseGuardian is within the threat model, consider removing the pauseGuardian's ability to change the `pendingPauseGuardian` address. Alternatively, consider replacing the offer/accept pattern with an admin-only function that directly sets the `pauseGuardian` address.

Update: *The compound team has resolved this issue by replacing* `_setPendingPauseGuardian` *and* `_acceptPauseGuardian` *with a single function,* `_setPauseGuardian`*, which allows only the admin to transfer the role of pauseGuardian to a different address.*

Low

Superfluous code in conditional

The conditional statement on line 997 of Comptroller.sol contains the code `if (msg.sender != pendingPauseGuardian || msg.sender == address(0))`. It is infeasible for `msg.sender` to ever be equal to `address(0)`, so the second half of the antecedent is superfluous. Consider simplifying this to `if (msg.sender != pendingPauseGuardian)`.

Similarly, on line 60 of Unitroller.sol, there is the conditional statement that begins: `if (msg.sender != pendingComptrollerImplementation || pendingComptrollerImplementation == address(0))`. When the second half of the of the antecedent is true (that is, when `pendingComptrollerImplementation == address(0)`), the first half must also be true (because `msg.sender` is *never* `address(0)`). Therefore, the second half of the antecedent is superfluous. Consider simplifying the statement to the logically equivalent version: `if (msg.sender != pendingComptrollerImplementation)`.

Notes

The pauseGuardian can pause part of a critical incentive mechanism

This patch aims to put critical system changes (i.e., changing the price oracle, close factor, collateral factor, liquidation incentive, etc.) behind a timelock, so users have *time* to exit (or "opt-out" of the upgrade) before a critical change takes place. It aims to make the two functions users may need to exit Compound (namely `redeem` and `repayBorrow`) always available to users, so users have the *ability* to exit before a critical change takes place. It also aims to make some functionality instantly pausable by the pauseGuardian, to give the pauseGuardian the ability to lock down some functionality in case a critical issue arises.

For the most part, these three sets of functionalities do not overlap. A notable exception is the ability to liquidate.

On the one hand, it is reasonable to believe that the ability to pause liquidation would be useful for preventing the exploitation of certain classes of possible vulnerabilities. So it is reasonable to want to give the pauseGuardian the ability to pause liquidation.

On the other hand, the ability to liquidate is a critical component of the mechanism used to keep Compound solvent. So it is also reasonable to want to keep this functionality safe from a potentially-malicious pauseGuardian.

Pausing liquidation can allow a malicious pauseGuardian (or those colluding with it) to avoid being liquidated. This could enable a malicious pauseGuardian to make high-risk/high-reward bets and

If a malicious pauseGuardian is within the scope of the threat model, consider whether the risk of a malicious pauseGuardian abusing their power to pause liquidation is greater than the risk of not being able to prevent exploitation of a possible vulnerability that would steal funds via liquidation. If it is, then consider removing the pauseGuardian's ability to pause liquidation.

It is unsafe to use CTokens as an underlying asset for other CTokens

If the asset underlying a CToken is another CToken with the same Comptroller, then the Comptroller's pauseGuardian can block calls to `redeem` and `repayBorrow` by setting `transferGuardianPaused` to `true`. This would cause `redeem` to revert on `doTransferOut` and `repayBorrow` to revert on `doTransferIn`. This means the pauseGuardian could prevent users from exiting such markets.

Consider refraining from ever using CTokens as underlying assets for other CTokens.

We are aware that the Compound developers do not intend to ever use CTokens as underlying assets for other CTokens (for this reason, as well as others), so we classify this issue as "note" severity and raise it here only for the benefit of users and future developers.

Timelock contract can be mistakenly frozen

It is essential that the admin of the Timelock contract rigorously tests the `bytes memory data` parameter that is passed to the `queueTransaction` and `executeTransaction` functions before executing the transaction. If this parameter is malformed, the Timelock contract can become frozen. For example, a malformed `data` parameter could result in the setting of an unintentionally large `delay` (when calling the `setDelay` function) or an inaccessible `admin` address (when calling the `setAdmin` function). Both of these could result in the Timelock contract becoming unusable.

Consider adding an upper bound check on the `delay_` parameter of the `setDelay` function, and using an offer/accept pattern for transferring the Timelock admin via the `setAdmin` function. Otherwise, be sure to test the queued transactions rigorously before executing them on mainnet.

Update: *This issue has been addressed in commit* `681833a557a282fba5441b7d49edb05153bb28ec`. *The Compound team has added a*

Uncaught return value

The call to `addToMarketInternal` on line 351 of Comptroller.sol returns a `uint256` that represents an error code. This return value is uncaught/unused. If the intention is to revert on a non-zero error code, consider catching this return value and using a `require` statement to check that it is equal to zero.

Update: *This issue has been resolved in commit `681833a557a282fba5441b7d49edb05153bb28ec` with the addition of lines 349-351 in Comptroller.sol, which check the returned error value*

Conclusion

No critical or high severity issues were found. Considerations were raised to point out the tradeoffs being made with this patch. Some changes were proposed to follow best practices and reduce the potential attack surface.

Related Posts

Beefy Zap Audit

OpenBrush Contracts Library Security Review

Linea Bridge Audit

Security Audits

Security Audits

Security Audits