OpenZeppelin

# Compound Polygon Bridge Receiver Audit

OPENZEPPELIN SECURITY | FEBRUARY 17, 2023

Announcements Security Audits

This security assessment was prepared by **OpenZeppelin**.

## Table of Contents

## Summary

Languages
Solidity

Total Issues
14 (4 resolved, 2 partially resolved)
Critical Severity Issues
0 (0 resolved)
High Severity Issues
0 (0 resolved)
Medium Severity Issues
2 (1 resolved)
Low Severity Issues
7 (3 resolved)
Notes & Additional Information
5 (0 resolved, 2 partially resolved)

## Scope

We audited the `compound-finance/comet` repository at the `2eb33b5e8454dba148373b6cb64ede4f7436fad7` commit.

In scope were the following contracts:

```
- contracts/bridges/BaseBridgeReceiver.sol
- contracts/bridges/polygon/PolygonBridgeReceiver.sol
- contracts/bridges/vendor/fx-portal/contracts/FxChild.sol
```

## System Overview

Compound plans to deploy its v3 of the protocol into Polygon. This audit focused on the deployment of the communication infrastructure between the existing Compound Governor contract on Ethereum Mainnet and the Polygon Network.

In order for the Compound Governor contract from Ethereum Mainnet to communicate with the Polygon deployment, a proposal needs to be submitted, voted on, and passed in the usual manner. Then, it will simply be sent through the Polygon bridge to the `PolygonBridgeReceiver` contract. This contract will then process the message and relay it to Polygon's deployed `Timelock` contract.

The receiver and the timelock are interlinked but have different responsibilities, which allows either of them be replaced with a different contract in the future. The receiver is meant to receive messages from the bridge, ask the timelock to enqueue transactions, and begin the execution of a proposal. The timelock serves the purpose of ensuring transactions can only be executed during the correct execution period, managing the queue of transactions, and executing transactions correctly when called upon by the receiver.

## Security Considerations & Trust Assumptions

Due to the interlinked nature of the contracts, there are no special outside roles other than the Compound's Ethereum timelock, which is the only message sender able to send across the bridge to the receiver. Indeed, most of the actions in both contracts rely on their interlinking, specifying the other for their respective responsibilities. This mitigates the possibility of malicious actors altering Governor proposals, and reduces the attack surface.

However, as all messages go through the Polygon bridge, the project has to rely on its security model and the availability of the system when sending a proposal. This means that in cases where the bridge is not functioning, as it happened with the Heimdall layer in 2022, Compound will not have the tools to take corrective actions over their Polygon deployment.

improve the outcome of the market for some players.

Another consideration is that due to the lack of a mechanism to retry transactions on Polygon, and the lack of a return-to-L1 message implementation, the Governor in L1 may need to restart the whole proposal process in the event that the proposal could not be submitted successfully to Polygon's Timelock contract. This could take considerable time and delay initiatives beyond expectations.

Finally, the decentralization of the Polygon network is based on a council of just a few actors, meaning that the entire ecosystem deployed on that network could suffer from the attack of only 5 of those addresses.

## Medium Severity

### `BaseBridgeReceiver` can be rendered inoperable by incorrectly setting the `localTimelock`

The intricate relationship between the `BaseBridgeReceiver` and `TimeLock` contracts requires extra caution when updating the `localTimelock` in the `BaseBridgeReceiver`, as an incorrect update could render the entire contract inoperable. The following errors must be avoided:

- Setting the new `localTimelock` value to an address that does not implement the `Timelock` interface (this includes sending the zero address by mistake)
- Setting the `localTimelock` variable to an address that implements the `Timelock` interface, but with an `admin` (or `pendingAdmin`) variable that is not set to the address of the `BaseBridgeReceiver` contract

If either of these two events were to happen, the `BaseBridgeReceiver` contract would be unable to successfully send any new messages to the `Timelock` contract and, as a result, would not be able to update its own `localTimelock` variable to address this issue.

Consider checking that the new `Timelock` address implements the `ITimelock` interface in the `initialize` and `setLocalTimelock` functions. Also consider checking that the `Timelock` contract's `admin` (or `pendingAdmin`) variable is set to the

***Update:*** *Resolved in pull request 665 at commit 1ffc7e9.*

## Proposals cannot be canceled

A proposal can queue its transactions in the `Timelock` contract by processing the message through the `BaseBridgeReceiver` contract.

However, even though the `Timelock` contract allows the `admin` address (in this case, the `BaseBridgeReceiver` contract) to cancel a certain transaction, the `BaseBridgeReceiver` contract does not implement the functionality to call that method.

This means that if a transaction needs to be canceled, a new proposal would need to be passed to change the `admin` address to an EOA or a contract that has the ability to cancel transactions. After the proposal is passed, it would need to be executed, and then the problematic transaction can finally be manually canceled.

Moreover, as the `executeProposal` function from the `BaseBridgeReceiver` contract is not access-controlled, any user may notice that a faulty or malicious transaction is queued and ready to be executed and the protocol will not have the option to stop them from executing it.

Consider implementing the functionality to cancel a transaction in the `BaseBridgeReceiver` contract.

***Update:*** *Acknowledged, not resolved. The Compound team stated:*

> *L2Timelock proposals cannot be canceled by design.*
>
> *Transactions begin as a proposal on L1. The L1 proposal is placed in the L1 Timelock (where it can be canceled), is queued for a period of time, and then is executed.*
>
> *The execution of the L1 proposal results in the transaction being enqueued on the L2. It is in a pending state for a period of time, but as far as governance is concerned it is as though the proposal has already been executed.*
>
> *We accept that once a transaction is enqueued on the L2, there is no way to cancel it.*

> maliciously.

## Low Severity

### Reversions in Polygon will not be seen in Ethereum

The protocol implements the functionality to send sensitive governor actions from Ethereum to the Polygon network by using its bridge.

However, due to the unidirectional path of such instructions, the mainnet governor will always assume that orders sent to Polygon were submitted correctly. This means that in case of a reverted proposal when processing the message, the governor will not be aware of it and it will not allow to retry the same proposal without going through the entire lifecycle again.

Consider handling failed submissions of proposals and completing the loop back from Polygon to Ethereum to allow retriable proposals.

***Update:** Acknowledged, not resolved. The Compound team stated:*

> *Comet's approach to L2s is unidirectional by design. Creating an omnidirectional loop would increase complexity and potentially create additional vectors for malicious activity.*

### Inconsistent transaction expiry

Within the `executeTransaction` function from the `Timelock` contract, a transaction is considered valid and will be executed as long as the transaction block's timestamp is *less than or equal* to `eta + GRACE_PERIOD`. However, the `BaseBridgeReciever` contract only considers a transaction valid if the transaction block's timestamp is instead *strictly less than* that same calculation.

Consider updating the contracts' logic to agree on the expiration of a transaction.

***Update:** Resolved in pull request 666 at commit fcb9ef3.*

### Inconsistent usage of `uint` across loops

However, when executing proposals, the `for` loop utilizes a `uint` variable type.

Even though the index of the `for` loop is not protected against overflow, in the unlikely case that more than 256 transactions are sent in a single proposal, the transaction would revert as there would be an identical transaction already queued.

Consider using the same type in both loops for consistency, or documenting the reason for having different implicit loop bounds. Additionally, consider removing the `unchecked` statement to reduce the attack vector when overflowing.

***Update:*** *Resolved in pull request 666 at commit 55712fb.*

## Lack of indexed parameter

Within `BaseBridgeReceiver.sol`, line 20 does not have the event parameters indexed.

Consider indexing event parameters to improve the off-chain services' ability to search and filter for specific events.

***Update:*** *Resolved in pull request 666 at commit 4b9bcb5.*

## Missing distinction between queued and ready-to-execute state

In the `BaseBridgeReceiver` contract, the `state` function returns the state of a given proposal. Currently, the possible state values are: queued, expired, or executed.

However, it is important to distinguish between transactions that are still in queue and require more time before being executed, and those that can already be executed.

This is especially important as the `executeProposal` function checks that the proposal is in the `queued` state, but does not check if the proposal can already be executed. As a result, some transactions that are queued but not ready to be executed will pass the initial check, and the transaction will only revert once the `executeTransaction` function from the `Timelock` contract is called.

***Update:*** *Acknowledged, not resolved. The Compound team stated:*

> *The net benefit of this change would simply be that the error thrown would be thrown by the BaseBridgeReceiver instead of being thrown by the Timelock. These two approaches seem functionally equivalent; we will not make this update.*

### Identical transactions can be executed inside the same proposal

When a proposal is sent through the Polygon bridge, the `BaseBridgeReceiver` contract processes it by queuing all transactions that are included in the proposal to the `Timelock` contract.

The `Timelock` contract validates that each transaction is unique by using the hash of the parameters of the transaction as the identifier. However, it is possible to send a duplicated transaction because of how the `signature` parameter of the function signature is included in the `data` parameter.

This is caused by how the method `abi.encode` is used when queuing the transactions. It is possible to send an identical transaction twice by sending the first transaction with the function `signature` and a second one with the 4 bytes function selector attached to the `data` field. This would create two different hashes, which causes the requirement to be satisfied when the proposal is being executed, and as a result the same transaction would be executed twice.

When queuing transactions, consider using the packed `callData` output instead of relying on the `signature` and `data` fields for getting the `txHash`.

***Update:*** *Acknowledged, not resolved. The Compound team stated:*

> *Executing identical transactions isn't a problem for us (there's nothing especially malicious that could be done by executing the same transaction twice. In fact, it's a leftover limitation of the original Timelock implementation that you're unable to take the same action multiple times (though there isn't an obvious use case to do so). We are hoping to keep the L2 Timelock contracts as similar as possible to the version deployed to mainnet, so we will leave this code as it is.*

contains the output from the `target.call` call.

However, the return value is not used in the `executeProposal` function from the `BaseBridgeReceiver` contract. This makes it impossible to access the return value of a transaction that is executed by the `BaseBridgeReceiver` contract, which may be useful for debugging erratic or unintended behaviors.

Consider retrieving these outputs from the `executeTransaction` function calls in the `executeProposal` function.

***Update:*** *Acknowledged, not resolved. The Compound team stated:*

> *It's unclear if exposing these return values would actually improve our ability to debug or monitor execution of transactions, especially given that reverted transactions won't emit any events or return any values. We will not make an update related to this recommendation.*

## Notes & Additional Information

### Inconsistent Nomenclature

There are a few instances where item names are inconsistent:

- `FXChild.sol` contains an interface named `IFxMessageProcessor`. Consider matching the name of the interface with the file's name.
- The `processMessageFromRoot` function from the `IFxMessageProcessor` interface has a parameter named `rootMessageSender`. This interface is implemented in the `PolygonBridgeReceiver` contract but the parameter's name has changed to `messageSender`. Consider being consistent when implementing the functionality.
- Within the `BaseBridgeReceiver` contract, if a proposal is expired, an error is thrown labeled as `ProposalNotQueued`. The name of this error could be misleading to a viewer. Consider changing the name of the event to reflect the current behavior, or even better, consider adding a new event to reflect such status.
- The `Timelock` allows the execution of a proposal after a certain delay but before a certain expiry time. This window is labeled as a `GRACE_PERIOD`, which is usually used to describe

***Update:*** *Partially resolved in pull request 666 at commit ccba0d9. The Compound team stated:*

> `GRACE_PERIOD`: *Will not update; we want to keep the Timelock interface as it is.*

## Missing docstrings

The `Timelock`, `ITimelock`, `FxChild`, and `PolygonBridgeReceiver` contracts are missing documentation for their functions.

A lack of documentation hinders reviewers' understanding of the code's intention which is fundamental to accurately assess not only security but also correctness. Additionally, docstrings improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned and the events emitted.

Consider thoroughly documenting all functions (and their parameters) that are part of the contracts' public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

***Update:*** *Partially resolved in pull request 666 at commit ccba0d9. The Compound team stated:*

> *Timelock – No change; we want to keep Timelock as similar as possible to the existing implementation*
> *FxChild – Will not add docstrings; this is a vendor contract.*

## Non-explicit imports are used

The use of non-explicit imports in the codebase can decrease the clarity of the code, and may create naming conflicts between locally defined and imported variables. This is particularly relevant when multiple contracts exist within the same Solidity files or when inheritance chains are long.

Throughout the codebase, global imports are being used. For instance:

- line 4 of `BaseBridgeReceiver.sol`

Following the principle that clearer code is better code, consider using named import syntax (`import {A, B, C} from "X"`) to explicitly declare which contracts are being imported.

***Update:*** *Acknowledged, not resolved. The Compound team stated:*

> *Our existing codebase uses non-explicit imports exclusively. We will continue to use non-explicit imports for consistency.*

### Same configuration values as Mainnet deployment

The Polygon deployment configuration is almost an exact copy of the Mainnet deployment configuration.

One difference is that the Polygon configuration contains USDT. In the past, Compound has refrained from using USDT as collateral, which causes an inconsistency across the protocol.

Moreover, during the first stages of a deployment, it is recommended to use conservative and stable values for the economic dynamics. Using the same values as the already established mainnet market could cause instabilities when using those from scratch.

Consider reviewing the configuration to ensure such values are correct for the Polygon deployment.

***Update:*** *Acknowledged, will resolve. The Compound team stated:*

> *Will update; the configuration values in #598 are not final.*

### Inconsistency between backlog and implementation

The pull request 598 contains a comment specifying that the deployment configuration should have a `supplyCap` value of `0`, while its current value is `500000e18`.

It seems the Compound Team is aware of this and will make this change prior to deployment.

Consider updating the `supplyCap` from the Polygon deployment configuration to zero.

***Update:*** *Acknowledged, will resolve. The Compound team stated:*

Two medium issues have been found. Some changes were proposed to follow best practices and reduce the potential attack surface.

## Appendix

### Monitoring Recommendations

While audits help in identifying potential security risks, the Compound team is encouraged to also incorporate automated monitoring of on-chain contract activity into their operations. Ongoing monitoring of deployed contracts helps in identifying potential threats and issues affecting the production environment.

Due to technical challenges in operating protocols across networks, we would suggest monitoring all events on Polygon to ensure that only expected activities are occurring. Particularly, consider monitoring:

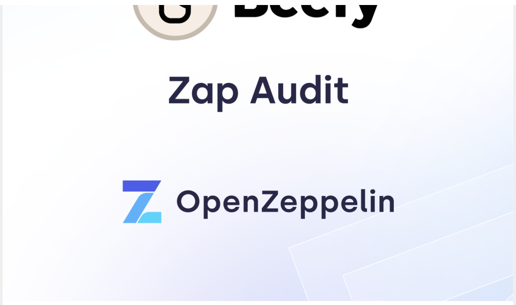
- The `ProposalCreated` event from the `BaseBridgeReciever` contract to ensure that transactions are correctly submitted through the bridge.
- The `ExecuteTransaction` event from the `Timelock` contract to ensure only expected transactions are being executed.
- The administrative events `NewLocalTimelock`, `NewGovTimelock`, `NewPendingAdmin`, `NewDelay`, and `NewAdmin` to ensure the correct ownership model is being handled.
- The addresses passed as input for the administrative changes during the proposal submission to prevent locking either of the contracts.
- The address calling the `executeProposal` function, searching for previous deployments or code inside that address that could affect the execution of the proposal.

## Related Posts

### Secure Implementations & Vulnerable Integrations in Smart Contracts: ERC-2771 Crisis Management

Dec '23 saw critical smart contract vulnerability, sparking a community-led response, exemplifying...

Announcements



### Beefy Zap Audit

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

Security Audits

### Announcing the OpenZeppelin Zero Knowledge Proofs (ZKP) Practice

With the rise of Zero-knowledge proofs as a prominent technology for blockchain privacy and...

Announcements