certora

Security Assessment

Compound Governor – Extended Pause

December 2025

Prepared for Compound

Table of content

certora

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Compound Extended Pause Feature (PR #249)	https://github.com/woof-software/comet/pull/249	3d059f2 (original commit)	Solidity

Project Overview

This document describes the manual code review findings of Compound Comet PR #249 - Extended Pause Feature. The following contracts are included in our scope:

contracts/CometCore.sol

contracts/CometStorage.sol

contracts/CometExt.sol

contracts/CometExtInterface.sol

contracts/CometMainInterface.sol

contracts/CometWithExtendedAssetList.sol

contracts/CometProxyAdmin.sol

The work was undertaken from December 10 - December 15, 2025. During this time, Certora's security researchers performed a manual audit of the above listed contracts introduced or modified by PR #249 and discovered several bugs in the codebase, which are summarized in the subsequent section.

Protocol Overview

Compound Comet is a DeFi lending protocol that allows users to supply collateral assets and borrow a base token. PR #249 introduces an Extended Pause mechanism that provides fine-grained, role-gated pause controls for selective disabling of specific user flows (supply, withdraw, transfer) without halting the entire market. This feature was designed to address scenarios such as price feed failures or asset delisting incidents, allowing governance to pause individual collateral assets (up to 24 assets) or specific operations (lenders vs borrowers) independently.

Threat and Security Overview

The Extended Pause feature adds governance-controlled circuit breakers using bitmap-based pause flags (uint24) to selectively disable supply, withdraw, and transfer operations per-asset or per-user-type (lenders vs borrowers).

Access control is enforced via the `onlyGovernorOrPauseGuardian` modifier, with idempotency checks preventing redundant state changes.

Manual Review Focus: Bitmap manipulation logic for 24 asset indices, enforcement layer integration in core functions (`supplyInternal`, `withdrawBase`, `withdrawCollateral`, `transferBase`, `transferCollateral`), consistency between pause control (`CometExt.sol`) and enforcement checks (`CometWithExtendedAssetList.sol`), and storage layout safety for proxy upgrades.

Key Findings:
One high-severity issue (H-01): type truncation in per-asset pause functions prevents unpausing assets with index ≥ 8, creating permanent lock conditions.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	0	-	-
High	1	1	1
Medium	0	-	-
Low	0	-	-
Informational	3	2	2
Total	4	3	3

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

Detailed Findings

ID	Title	Severity	Status
H-01	Type conversion truncation prevents unpausing collateral assets with index ≥ 8	High	Fixed.
I-01	Type truncation in extended pause view functions	Informational	Fixed.
I-02	Pause check after token transfer	Informational	Fixed.
I-03	Missing pause flag for pausing lender deposits while allowing loan repayments	Informational	Acknowledged.

High Severity Issues

H-01. Type conversion truncation prevents unpausing collateral assets with index ≥ 8

Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: CometExt.sol	Status: Fixed.	

Description:

A type conversion vulnerability exists in `CometExt.sol` that prevents the unpausing of collateral assets with `index ≥ 8`. The vulnerability affects three pause control functions that govern core protocol operations for individual collateral assets:

1. `pauseCollateralAssetWithdraw()`
2. `pauseCollateralAssetSupply()`
3. `pauseCollateralAssetTransfer()`

Each function contains a flawed status check before modifying the pause state:

```JavaScript
if (toBool(uint8(collateralsXxxPauseFlags & (uint24(1) << assetIndex))) == paused)
 revert CollateralAssetOffsetStatusAlreadySet(...);
```

When `assetIndex >= 8`, the bit mask value `(uint24(1) << assetIndex)` exceeds 255. For example, `1 << 8 = 256`.

The `uint8` cast silently truncates this to zero, breaking the logic:

- `Asset 8: (1 << 8) = 256 -> uint8(256) = 0`
- `Asset 9: (1 << 9) = 512 -> uint8(512) = 0`

This creates an asymmetric behavior where:

- Pausing works: `toBool(uint8(0)) == true -> false == true -> no revert`
- Unpausing fails: `toBool(uint8(256)) == false -> false == false -> reverts permanently`

`CometExt` is called via `delegatecall` from `CometWithExtendedAssetList` through its fallback function. When assets 8-23 are paused, they become permanently locked because the unpause operation always reverts. This blocks three critical operations in `CometWithExtendedAssetList`, depending on which of the functionalities for specific assets are paused:

1. withdrawCollateral() - Users cannot withdraw collateral
2. supplyCollateral() - Users cannot add collateral to prevent liquidations
3. transferCollateral() - Users cannot transfer collateral between accounts

Once a collateral asset with `index ≥ 8` is paused, it becomes permanently locked with no recovery mechanism short of a contract upgrade.

Recommendations:

Remove the `uint8` cast from all three functions.

```
JavaScript
- if (toBool(uint8(collateralsWithdrawPauseFlags & (uint24(1) << assetIndex))) == paused)

+ if (((collateralsWithdrawPauseFlags & (uint24(1) << assetIndex)) != 0) == paused)
```

Apply the same fix to `pauseCollateralAssetSupply()` and `pauseCollateralAssetTransfer()`.
Customer's response: Fixed in 6e2d418

Fix Review: Fix confirmed.

Informational Issues

I-01.Type truncation in extended pause view functions

Description:

View functions in `CometWithExtendedAssetList.sol` use `uint8` casts when checking `extendedPauseFlags` (declared as `uint24`), limiting implementation to 8 bits despite the type system indicating 24 bits are available.

Affected: isCollateralWithdrawPaused(), isCollateralSupplyPaused(), isBaseSupplyPaused(), isLendersTransferPaused(), isBorrowersTransferPaused(), currentPauseOffsetStatus()

```
JavaScript
   function isCollateralWithdrawPaused() public view returns (bool) {
    return toBool(uint8(extendedPauseFlags & (uint24(1) <<
PAUSE_COLLATERALS_WITHDRAW_OFFSET)));
    //             ^^^^^ Truncates to 8 bits
}
```

Currently safe: All 8 offsets (0-7) are within `uint8` range.

Future risk: If offsets 8-23 are added, checks silently return incorrect results:

```
JavaScript
uint24(1) << 8 = 256 → uint8(256) = 0 → toBool(0) = false
```

Recommendations:

Remove `uint8` casts:

```JavaScript
function isCollateralWithdrawPaused() public view returns (bool) {
-   return toBool(uint8(extendedPauseFlags & (uint24(1) <<
PAUSE_COLLATERALS_WITHDRAW_OFFSET)));
+   return (extendedPauseFlags & (uint24(1) << PAUSE_COLLATERALS_WITHDRAW_OFFSET)) != 0;
}
```

Customer's response: Fixed in c1e706a & 41461c0

Fix Review: Fix confirmed.

I-02. Pause check after token transfer

Description:

In the `supplyCollateral()` function, the per-asset pause check is performed AFTER the ERC-20 token transfer, rather than before. While the transaction correctly reverts if paused, this ordering creates unnecessary gas waste for users.

```JavaScript
   function supplyCollateral(address from, address dst, address asset, uint128 amount)
internal {
    amount = safe128(doTransferIn(asset, from, amount));

    AssetInfo memory assetInfo = getAssetInfoByAddress(asset);
    uint8 offset = assetInfo.offset;

@>  if (isCollateralAssetSupplyPaused(offset)) revert CollateralAssetSupplyPaused(offset);
}
```

Other pause enforcement functions correctly check pause status BEFORE external calls:

```JavaScript
function withdrawCollateral(...) internal {
    AssetInfo memory assetInfo = getAssetInfoByAddress(asset);
    uint8 offset = assetInfo.offset;

@>    if (isCollateralAssetWithdrawPaused(offset)) revert
CollateralAssetWithdrawPaused(offset);


    doTransferOut(asset, to, amount);
}
```

Recommendations:

Move the pause check before `doTransferIn()`:

```JavaScript
function supplyCollateral(address from, address dst, address asset, uint128 amount) internal
{
+   AssetInfo memory assetInfo = getAssetInfoByAddress(asset);
+   uint8 offset = assetInfo.offset;
+
+   if (isCollateralAssetSupplyPaused(offset)) revert CollateralAssetSupplyPaused(offset);
+
    amount = safe128(doTransferIn(asset, from, amount));

-   AssetInfo memory assetInfo = getAssetInfoByAddress(asset);
-   uint8 offset = assetInfo.offset;
-
-   if (isCollateralAssetSupplyPaused(offset)) revert CollateralAssetSupplyPaused(offset);

    TotalsCollateral memory totals = totalsCollateral[asset];
    ...
}
```

Customer's response: Fixed in 1b39615

Fix Review: Fix confirmed.

I-03. Missing pause flag for pausing lender deposits while allowing loan repayments

Description:

The extended pause system provides separate pause controls for lenders and borrowers in withdrawal and transfer operations, but not for supply. The `isBaseSupplyPaused()` flag blocks both lender deposits and borrower repayments without distinction.

This creates a problematic scenario during market stress: pausing new deposits to limit exposure also prevents borrowers from repaying their loans, when reducing debt is critical.

Recommendations:

Add separate pause flags for lender deposits and borrower repayments.

Customer's response: Acknowledged. This functionality was outside the scope of the audit. Additionally, we do not see a market scenario where pausing base asset lending while still allowing borrower repayments would be beneficial. Both supplying the base asset and repaying borrowed positions are core protocol operations and are intended to remain available at all times.

As a result, the current behavior of isBaseSupplyPaused(), which affects both deposits and repayments, is considered acceptable within the intended design and threat model of the protocol.

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.