Compound Gas Optimizations Audit

OPENZEPPELIN SECURITY | MARCH 6, 2020

Security Audits

The Compound team engaged us to audit part of their latest gas optimizations in the protocol. The audited patch is `1cf040412fc8a0cfd1187bb0d8a525c3ae2f78aa`, and the scope was limited to the modifications applied in files:

- `CDaiDelegate.sol`
- `CErc20.sol`
- `CEther.sol`
- `Comptroller.sol`
- `ComptrollerG1.sol`
- `ComptrollerInterface.sol`
- `DAIInterestRateModel.sol`
- `JumpRateModel.sol`
- `Maximillion.sol`
- `SafeMath.sol`
- `WhitepaperInterestRateModel.sol`
- `CToken.sol` (with the caveat mentioned below)

It must be highlighted that we left out of scope:

- All changes to the `PriceOracle.sol`, `SimplePriceOracle.sol` and `PriceOracleProxy.sol` files

No major issues were found during our assessment. We do make some suggestions to improve the code's overall quality and the system's efficiency in terms of gas costs.

High-level overview of the changes

The audited patch consists of a series of gas optimizations to reduce operation costs, without significantly modifying the system's behavior.

Some of the most relevant changes are the removal of unused verification hooks in the `CToken` contract, a better use of local variables to reduce the amount of `SLOAD`s operations, and the removal of redundant transfer checks to make error-handling more efficient.

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

Following we list our recommendations, in order of importance.

Critical severity

None.

High severity

None.

Medium severity

None.

Low severity

[L01] Function redeemFresh may not fail early

The `redeemFresh` function of the `CToken` contract has been modified to inline a validation previously done in the `redeemVerify` function of the `Comptroller` contract. This

applied to the contract's storage and events emitted. This should not only favor the function's readability, but also save gas costs for failed transactions.

Notes & Additional Information

[N01] Unnecessary reads from storage

The `accrueInterest` function of the `CToken` contract was modified to, among other things, reduce the amount of storage read operations. This is intended to make the system more efficient in terms of gas costs. However, there still are redundant read operations that can be avoided. In particular:

- State variable `totalBorrows` is assigned to the `borrowsPrior` local variable in line 394, but later is read from storage in lines 426 and 431
- State variable `totalReserves` is assigned to the `reservesPrior` local variable in line 395, but later is read from storage in line 436

Consider reusing the corresponding local variables to avoid redundant `SLOAD`s operations.

[N02] Inconsistent use of enum for error messages

A verification previously done in the `redeemVerify` function has been inlined at the end of the `redeemFresh` function to avoid an unnecessary call to the `Comptroller` contract. However, the error message included in the new `require` statement has been hardcoded instead of using the `FailureInfo` enum used throughout the rest of the function.

Consider adding the new error to the `FailureInfo` enum of the `TokenErrorReporter` contract so as to keep a consistent style in error messages of the `redeemFresh` function.

[N03] Inconsistent use of warning comments when calling doTransferIn function

After the removal of the internal calls to the `checkTransferIn` function in the `mintFresh`, `repayBorrowFresh` and `_addReservesFresh` functions of the `CToken` contract, warning comments explaining the behavior of the `doTransferIn` function become more

To be consistent in the use of warning comments and favor the code's readability, consider adding similar warnings in the `_addReservesFresh` function before `doTransferIn` is called.

[N04] Unused verification functions

Consider removing the `transferVerify`, `mintVerify`, `redeemVerify`, `borrowVerify`, `repayBorrowVerify`, `seizeVerify` functions, since they are no longer being used. Alternatively, consider adding an inline comment explicitly stating that these functions are never used to favor readability.

This note should be disregarded if the mentioned external functions are used by off-chain clients.

[N05] Unused error codes

Consider removing error codes `REPAY_BORROW_TRANSFER_IN_NOT_POSSIBLE`, `TOKEN_INSUFFICIENT_ALLOWANCE`, `TOKEN_INSUFFICIENT_BALANCE`, `MINT_TRANSFER_IN_NOT_POSSIBLE`, `REPAY_BORROW_TRANSFER_IN_NOT_POSSIBLE`, `ADD_RESERVES_TRANSFER_IN_NOT_POSSIBLE` from the `FailureInfo` enum of the `TokenErrorReporter` contract, since they are no longer being used. Once the error codes are removed from the contract, consider also removing them from the `scenario/src/ErrorReporterConstants.ts` file.

Conclusions

No critical nor high issues were found. Minor suggestions were made to improve the code's overall quality.

Related Posts

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