OpenZeppelin | security

Morpho Bundler V3 Audit

Morpho

January 2, 2025

Table of Contents

Summary

Type	DeFi
Timeline	From 2024-12-05 To 2024-12-11
Languages	Solidity

Total Issues	15 (0 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	2 (0 resolved)
Notes & Additional Information	12 (0 resolved)
Client Reported Issues	1 (0 resolved)

Scope

We conducted a diff audit of the morpho-org/bundler-v3 repository at commit d7940c9 against the morpho-org/morpho-blue-bundlers repository at commit 1fa1725.

In scope of the diff audit were the following files:

```
src
├── Bundler.sol
├── adapters
│   ├── CoreAdapter.sol
│   ├── EthereumGeneralAdapter1.sol
│   ├── GeneralAdapter1.sol
│   └── migration
│       ├── AaveV2MigrationAdapter.sol
│       ├── AaveV3MigrationAdapter.sol
│       ├── AaveV3OptimizerMigrationAdapter.sol
│       ├── CompoundV2MigrationAdapter.sol
│       └── CompoundV3MigrationAdapter.sol
├── interfaces
│   ├── IAaveV2.sol
│   ├── IAaveV3.sol
│   ├── IAaveV3Optimizer.sol
│   ├── IBundler.sol
│   ├── ICEth.sol
│   ├── ICToken.sol
│   ├── ICompoundV3.sol
│   ├── IComptroller.sol
│   ├── IDaiPermit.sol
│   ├── IStEth.sol
│   ├── IWNative.sol
│   └── IWstEth.sol
└── libraries
    ├── BytesLib.sol
    ├── ErrorsLib.sol
    ├── MathRayLib.sol
    └── UtilsLib.sol
```

However, the following new files were fully audited:

```
src
├── adapters
│   └── ParaswapAdapter.sol
└── interfaces
    ├── IParaswapAdapter.sol
    └── IAugustusRegistry.sol
```

Update: *The most recent commit hash examined after the fixes was commit* `6b24812`.

System Overview

Morpho is a lending protocol developed by Morpho Labs. In addition to building the protocol, the Morpho Labs team has developed a set of periphery Bundler contracts that allow EOAs to batch actions in one transaction. Examples of some of these actions are: interacting with the Morpho protocol like repaying debt, borrowing and supplying collateral, wrapping and unwrapping ETH, and migrating positions from other lending protocols like Compound and Aave.

The earlier version of the Bundler contract used a self-`delegatecall` method to call the functions defined in the various parent contracts of the Bundler. Since the Bundler contract held all the allowances of users' funds, it could not call arbitrary addresses. The new Bundler contract system has decoupled call-dispatching and approval management. Now, the Bundler contract only acts as a dispatcher and calls different adapters that have allowance from users and do actions on their behalf. This allows the Bundler to be able to call arbitrary addresses with arbitrary calldata as it does not hold any funds or approvals. This also makes adding new functions to the Bundler easier as one would only need to deploy a new adapter without risking the existing approved user funds.

Security Model and Trust Assumptions

It is assumed that the user (or the bundle creator) crafts the calldata correctly and only calls trusted contracts through the Bundler.

Low Severity

L-01 Missing Checks Potentially Leads to Leftover Funds in Contract

In `ParaswapAdapter`, the `sell` function is used to sell an exact amount of `srcToken`. It can also check for a minimum purchased amount of `destToken`. Its complementary function, the `buy` function, is used to buy an exact amount of `destToken` and can also check for a maximum sold amount of `srcToken`. Both of these functions leverage the `swap` function [1] [2] to execute the defined swap using an `augustus` contract.

However, for the `sell` function, the `swap` function will only guarantee that the `srcAmount` sold is either less than or equal to the `maxSrcAmount` but not strictly equal, meaning the function does not guarantee the exact amount sold. Similarly, for the `buy` function, the `swap` function will only guarantee that the `destAmount` bought is either bigger than or equal to the `minDestAmount` but not strictly equal, meaning the function does not guarantee the exact amount bought.

For example, if a user sends an amount of `srcToken` to the contract and wants to sell all the balance they transferred, they might expect that at the end of the transaction, the `srcToken` balance of the `ParaswapAdapter` is zero, skipping the sweeping action. However, if the underlying `augustus` contract misbehaves, passing the checks in the `swap` function, it is possible that some `srcToken` balance is left in the contract, leaving an opportunity for a malicious actor to back-run the transaction and sweep the tokens after the transaction. A similar behavior is allowed when buying tokens.

Consider adjusting the balance checks to the exact amounts specified when selling or buying, respectively. Doing this will help prevent unexpected remaining funds in the `ParaswapAdapter` contract. Alternatively, consider removing the word `exact` from the docstrings in the adapter.

L-02 Missing Docstrings

In `GeneralAdapter1.sol`, the `onMorphoSupply`, `onMorphoSupplyCollateral`, `onMorphoRepay`, and `onMorphoFlashLoan` functions, as well as the `swap` function in `ParaswapAdapter.sol` are missing docstrings.

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's 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).

Notes & Additional Information

N-01 Floating Pragma

Pragma directives should be fixed to clearly identify the Solidity version with which the contracts will be compiled. The `CoreAdapter.sol` file has the `solidity ^0.8.0` floating pragma directive.

Consider using fixed pragma directives.

N-02 Incorrect Security Contact Tag

Throughout the codebase, multiple instances of the security contact tag being used incorrectly were identified.

Consider replacing all occurrences of `@custom:contact` with the recommended `@custom:security-contact` tag as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

N-03 Unnecessary Cast

Within the `ParaswapAdapter` contract, the `address(augustus)` cast is unnecessary.

To improve the overall clarity and intent of the codebase, consider removing any unnecessary casts.

N-04 Unused Imports

In `GeneralAdapter1.sol`, multiple instances of unused imports were identified.

- `IAllowanceTransfer` in line 5
- `IERC20Permit` in line 7
- `Signature` and `Authorization` in line 8.

Consider removing unused imports to improve the overall clarity and readability of the codebase.

N-05 Misleading Error

When executing a `swap` in the `ParaswapAdapter` contract, the function will check whether the passed `augustus` address is valid and revert with the `AugustusNotInRegistry` custom error if the address is invalid. This suggests that the passed `augustus` address is not in the registry. However, the `isAugustusValid` function in the `AugustusRegistry` contract will return `false` if the address is not in the registry or is banned.

Consider renaming the custom error to `InvalidAugustus` to better reflect validation failure.

N-06 Unused State Variable

Within the `EthereumGeneralAdapter1` contract, the `DAI` state variable is unused.

To improve the overall clarity and intent of the codebase, consider removing any unused state variables.

N-07 Function Visibility Overly Permissive

Throughout the codebase, multiple instances of functions having unnecessarily permissive visibility were identified:

- The `morphoCallback` function in `GeneralAdapter1.sol` with `internal` visibility could be limited to `private`.
- The `swap` function in `ParaswapAdapter.sol` with `internal` visibility could be limited to `private`.
- The `updateAmounts` function in `ParaswapAdapter.sol` with `internal` visibility could be limited to `private`.

To better convey the intended use of functions, consider changing a function's visibility to be only as permissive as required.

N-08 Unnecessary Check

The `swap` function of the `ParaswapAdapter` contract executes a swap with the defined `augustus` contract. At the end of the swap, the function transfers the remaining `destAmount` of the `destToken` to the defined `receiver` if the `destAmount` is larger than zero and the receiver is not the address of the `ParaswapAdapter` contract.

However, the `minDestAmount` is guaranteed to be larger than zero and `destAmount` can only be either equal or greater than `minDestAmount`. This means that the `destAmount` is guaranteed to be larger than zero, making the `destAmount > 0` check on line 171 unnecessary.

Consider removing the `destAmount > 0` check on line 171 to make the code more concise and save gas by avoiding unnecessary checks.

N-09 `ParaswapAdapter` Is Not Compatible With Older `Augustus` Versions

The `ParaswapAdapter` contract is an adapter that facilitates swaps on Paraswap's `Augustus` contract. `ParaswapAdapter` is built to work with the latest version (`v6`) of the `Augustus` contract. The `Augusutus` contract address is provided by the user when executing swaps. The user can, however, unknowingly provide the address of an older or a future/newer version of `Augustus` which can result in unwanted scenarios. This is because `ParaswapAdapter` is not necessarily compatible with other versions of `Augustus`.

Consider adding a docstring stating the `Augustus` version which the `ParaswapAdapter` is designed to interact with.

N-10 Misleading Documentation

Throughout the codebase, multiple instances of misleading documentation were identified:

- Both comments in lines 45 and 60 for the `nativeTransfer` and `erc20Transfer` functions suggest that an amount of zero can be transferred. However, both functions require the amount to be different from zero [1] [2].

- The comment in line 75 of the `ParaswapAdapter` contract suggests that `callData` can change if `marketParams.loanToken == destToken`, which does not occur inside the function.

Consider updating the above-mentioned comments to improve the clarity of the codebase.

In addition, the comment in line 108 of the `ParaswapAdapter` contract implies that the `buyMorphoDept` function buys an amount corresponding to a user's Morpho debt. However, in case the `debtAmount` is zero, the function will still execute the `buy` function using the values inside the `callData`.

Consider either returning early if the user has no `debtAmount` or updating the function's documentation accordingly.

N-11 Swapped Values During Variable Assignment

In the `CommonTest` contract, the inputs of the `makeAddr` function assigned to the `SUPPLIER` and `OWNER` variables are swapped.

Consider assigning the expected value to the corresponding variable to enhance code clarity and avoid confusion.

N-12 Inconsistency in Usage of the `onlyBundler` Modifier

The usage of the `onlyBundler` modifier is inconsistent across adapters. The `CoreAdapter` contract defines the `onlyBundler` modifier, which restricts function access to only the `Bundler` contract. Most adapters inherit from `CoreAdapter` and apply this modifier to all their functions.

Some functions genuinely need this protection. For example, `erc20TransferFrom` requires it to prevent unauthorized token transfers via reentrancy. Other functions do not need this protection. For example, `wrapNative` is naturally secure since it requires ETH to be sent first. The `ParaswapAdapter` contract is the only adapter that does not use the modifier on its functions at all.

To improve code consistency and clarity, consider either: adding the `onlyBundler` modifier to `ParaswapAdapter` functions, or removing the modifier from functions that don't require this protection. This would make the security model more explicit and easier to understand.

Client Reported

CR-01 Possible Reentrancy Vulnerability

When executing an external call inside the `Bundler` contract, the immediate callee is allowed to reenter the `Bundler` contract and execute a new bundle of calls. In case the user calls a compromised callee, the latter can reenter the `Bundler` contract and execute unauthorized arbitrary operations on other adapters within the initial `initiator` context, potentially retrieving funds on behalf of the user and sending them to another address they control.

To mitigate this risk, consider implementing a mechanism that determines whether a target contract can perform reentrant calls and the calldata with which they can re-enter.

Conclusion

After reviewing the architecture and changes in the new Bundler system, we can attest to its overall quality and robustness, and are glad to see security best practices implemented. We found the codebase to be highly readable, well-modularized, and sufficiently documented. Morpho Labs team was very responsive throughout the engagement and helped us quickly understand the codebase.