Morpho Bundler v3 Security Review

Auditors

Eric Wang, Lead Security Researcher

MiloTruck, Lead Security Researcher

Om Parikh, Security Researcher

Rvierdiiev, Associate Security Researcher

Report prepared by: Lucas Goiriz

January 2, 2025

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

The Morpho protocol is a decentralized, noncustodial lending protocol implemented for the Ethereum Virtual Machine. The protocol had two main steps in its evolution with two independent versions: Morpho Optimizers and Morpho Blue.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Morpho Bundler v3 according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 6 days in total, Morpho engaged with Spearbit to review the bundler-v3 protocol. In this period of time a total of **7** issues were found.

Summary

Project Name	Morpho
Repository	bundler-v3
Commit	5359c38c
Type of Project	DeFi, AMM
Audit Timeline	Dec 9th to Dec 15th
Fix period	Dec 15

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	3	2	1
Gas Optimizations	0	0	0
Informational	3	3	0
Total	**7**	**6**	**1**

5 Findings

5.1 Medium Risk

5.1.1 `GeneralAdapter1.morphoRepay()` incorrectly fetches the `borrowShares` of `initiator` instead of `onBehalf`

Severity: Medium Risk

Context: GeneralAdapter1.sol#L309-L312, GeneralAdapter1.sol#L316

Description: When `GeneralAdapter1.morphoRepay()` is called with `shares == type(uint256).max`, `MorphoLib.borrowShares()` is called to set `shares` to the amount of `borrowShares` held by the `initiator`:

```
if (shares == type(uint256).max) {
    shares = MorphoLib.borrowShares(MORPHO, marketParams.id(), initiator());
    require(shares != 0, ErrorsLib.ZeroAmount());
}
```

However, later on in the function, `MORPHO.repay()` is called to repay assets for `onBehalf` and not `initiator`:

```
(uint256 repaidAssets, uint256 repaidShares) = MORPHO.repay(marketParams, assets, shares, onBehalf,
↪  data);
```

As such, passing the `initiator` address to `MorphoLib.borrowShares()` as shown above is incorrect, since the `borrowShares` of `onBehalf` should be fetched instead. This causes an incorrect amount of shares to be repaid whenever `GeneralAdapter1.morphoRepay()` is called while `onBehalf` and `initiator` happen to be different addresses.

Recommendation: Modify the third parameter passed to `MorphoLib.borrowShares` to be `onBehalf`:

```
- shares = MorphoLib.borrowShares(MORPHO, marketParams.id(), initiator());
+ shares = MorphoLib.borrowShares(MORPHO, marketParams.id(), onBehalf);
```

Morpho: Fixed in commit 6e6ef769.

Spearbit: Verified, the recommended fix was implemented.

5.2 Low Risk

5.2.1 Lack of `onlyBundler` modifier inside functions of `ParaswapAdapter` adapter allows stealing funds

Severity: Low Risk

Context: ParaswapAdapter.sol#L55

Description: The `ParaswapAdapter` contract has several functions that are not protected by the `onlyBundler` modifier. Because of that, it's possible to trigger them from any caller. This creates a possibility of stealing user's funds from the adapter.

The purpose of the adapter is to receive funds, swap them and then transfer them or leave in the contract. There are few cases when funds are still in the adapter after the paraswap swap:

- Source token leftover.
- When `receiver` of the swap is `ParaswapAdapter` itself (see ParaswapAdapter.sol#L171).

In case any malicious contract gets a callback in the next step of bundle multicall after the swap, while funds are still in the `ParaswapAdapter`, it can swap them through the paraswap and transfer out of the `ParaswapAdapter`.

As result of this, when the victim expects to transfer tokens back from `ParaswapAdapter`(likely using `type(uint256).max` as `amount` to the `CoreAdapter.erc20Transfer()` call), the call won't revert as attacker just leaves 1 wei to bypass zero amount check (see CoreAdapter.sol#L70).

Recommendation: It's recommended to apply `onlyBundler` modifier to functions of `ParaswapAdapter`.

Morpho: Fixed in PR 192. The scenario assumes the user calls a hostile contract after using Paraswap, but before skimming. It is the user's responsibility not to use an adapter to call a hostile contract that can exploit that adapter's funds or approvals. However, we have still added the `onlyBundler` modifier to help reasoning across all non-callback external adapter functions.

Spearbit: Verified, the `onlyBundler` modifier has been added to `swap()`, which ensures all functions in the adapter can only be called by the bundler.

5.2.2 `EthereumGeneralAdapter1.wrapStEth()` leaves dust amounts of stETH behind due to Lido's 1-2 wei corner case

Severity: Low Risk

Context: CoreAdapter.sol#L68-L72, EthereumGeneralAdapter1.sol#L111-L115

Description: When `EthereumGeneralAdapter1.wrapStEth()` is called with `amount = type(uint256).max`, the stETH balance of the contract is passed to `WST_ETH.wrap()`:

```
if (amount == type(uint256).max) amount = IERC20(ST_ETH).balanceOf(address(this));

require(amount != 0, ErrorsLib.ZeroAmount());

uint256 received = IWstEth(WST_ETH).wrap(amount);
```

This is meant to wrap the contract's entire stETH balance into wstETH. Logically, this means that there should not be any stETH remaining in the contract. However, for certain stETH balances, passing `stETH.balanceOf()` to `wstETH.wrap()` directly could leave dust amounts of stETH behind due to the 1-2 wei corner case.

Under the hood:

- `stETH.balanceOf()` calls `stETH.getPooledEthByShares()` to convert the shares held by the contract into ETH.
- `wstETH.wrap()` → `stETH.transferFrom()` calls `stETH.getSharesByPooledEth()` to convert the amount passed into shares.

This results in the following math:

```
amount = getPooledEthByShares(shares) = shares * totalPooledEther / totalShares
sharesTransferred = getSharesByPooledEth(amount) = amount * totalShares / totalPooledEther
```

where:

- `shares` is the amount of stETH held by the contract in shares.
- `sharesTransferred` is the amount of shares transferred by `wstETH.wrap()`.

Due to rounding down, `sharesTransferred < shares` is possible, which ends up leaving a dust amount of stETH behind.

Subsequent calls in the same bundle might expect the stETH balance of the contract to be `0` after `wrapStEth()` is called (eg. a subsequent call also uses `stETH.balanceOf()` to determine the amount of that operation). However, since this is not the case, their execution could be affected.

Note that the same situation also occurs when `CoreAdapter.erc20Transfer()` is called with `amount = type(uint256).max` to transfer out the contract's stETH balance.

Proof of Concept: The following proof of concept demonstrates that 1 wei of stETH is left behind when `1e18` of stETH is transferred:

```
// SPDX-License-Identifier: GPL-2.0-or-later
pragma solidity ^0.8.0;

import "lib/forge-std/src/Test.sol";
import {ForkTest} from "test/fork/helpers/ForkTest.sol";
import {Bundler} from "src/Bundler.sol";
import {EthereumGeneralAdapter1, IERC20} from "src/adapters/EthereumGeneralAdapter1.sol";

contract StEthRoundingTest is ForkTest {
    function setUp() public override {
        // Deploy adapter and bundler
        bundler = new Bundler();
        ethereumGeneralAdapter1 = new EthereumGeneralAdapter1(
            address(bundler),
            address(0xBBBBBbbBBb9cC5e90e3b3Af64bdAF62C37EEFFCb),
            address(0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2),
            address(0x6B175474E89094C44Da98b954EedeAC495271d0F),
            address(0x7f39C581F595B53c5cb19bD0b3f8dA6c935E2Ca0),
            address(0x58D97B57BB95320F9a05dC918Aef65434969c2B2),
            address(0x9D03bb2092270648d7480049d0E58d2FcF0E5123)
        );
    }

    function testWrapStEth() public {
        IERC20 stETH = IERC20(ethereumGeneralAdapter1.ST_ETH());

        // Mint 1e18 of stETH to adapter
        deal(address(stETH), address(ethereumGeneralAdapter1), 1e18);
        assertEq(stETH.balanceOf(address(ethereumGeneralAdapter1)), 1e18 - 1); // -1 here is due to
        ↪ rounding

        // Call wrapStEth() with amount = uint256.max
        bundle.push(_wrapStEth(type(uint256).max, RECEIVER));
        bundler.multicall(bundle);

        // Adapter has 1 wei of stETH leftover
        assertEq(stETH.balanceOf(address(ethereumGeneralAdapter1)), 1);
    }
}
```

It can be run with:

```
forge test -vvv --match-path test/StEthRoundingTest.t.sol --fork-url $ETHEREUM_RPC_URL --block-number
↪ 21388419
```

Recommendation: Document this limitation in the natspec of `EthereumGeneralAdapter1.wrapStEth()` and `CoreAdapter.erc20Transfer()`.

Morpho: Acknowledged. Note that it is possible to directly wrap ETH to wstETH using wstETH's receive function.

Spearbit: Acknowledged.

5.2.3 `nativeTransfer()` and `erc20Transfer()` cannot be used to skim remaining balances due to zero amount check

Severity: Low Risk

Context: CoreAdapter.sol#L54, CoreAdapter.sol#L70

Description: In `CoreAdapter`, the `nativeTransfer()` and `erc20Transfer()` functions check that the `amount` to transfer out is non-zero, and revert if so:

```
require(amount != 0, ErrorsLib.ZeroAmount());
```

However, this could cause an entire bundle to revert if `nativeTransfer()` or `erc20Transfer()` is used to skim leftover balances at the end of a bundle. A user might wish to append a call to `nativeTransfer()`/`erc20Transfer()` with `amount = uint256.max` at the end of a bundle just in case there are any leftover tokens, but this check will cause the entire bundle to revert if there happens to be no tokens remaining.

A possible workaround would be to specify `skipRevert = true`, however, this makes it possible to skip a revert in `ERC20.transfer()` instead.

Recommendation: Consider skipping the transfer when `amount = 0` instead of reverting.

Morpho: Fixed in PR 197 to allow `amount = 0` on skimming.

Spearbit: Verified, `amount = 0` is now allowed when `nativeTransfer()`/`erc20Transfer()` is called with `amount = uint256.max`.

5.3 Informational

5.3.1 Minor issues with code and comments

Severity: Informational

Context: EthereumGeneralAdapter1.sol#L18, EthereumGeneralAdapter1.sol#L74, AaveV2MigrationAdapter.sol#L52-L54, AaveV3MigrationAdapter.sol#L53-L55, ParaswapAdapter.sol#L75, ParaswapAdapter.sol#L116, ParaswapAdapter.sol#L171, BytesLib.sol#L12-L14

Description/Recommendation:

1. EthereumGeneralAdapter1.sol#L18 - The immutable variable `DAI` is not used and can be removed.
2. EthereumGeneralAdapter1.sol#L74 - This check is present in the wrapper contract (Wrapper.sol#L53) and can be removed.
3. ParaswapAdapter.sol#L75 - This comment can be moved to the `buyMorphoDebt()` function, as it is the function with the `marketParams` parameter.
4. ParaswapAdapter.sol#L116 - This comment should be updated as the leftover `srcToken` will not be sent to the `receiver` at the end of the call.
5. ParaswapAdapter.sol#L171 - The check `destAmount > 0` can be removed. It always holds because `destAmount >= minDestAmount` and `minDestAmount != 0` are checked at the beginning of the `swap()` function.
6. BytesLib.sol#L12-L14 - This assembly block is missing the `memory-safe` annotation.
7. AaveV2MigrationAdapter.sol#L52-L54, AaveV3MigrationAdapter.sol#L53-L55 - The natspec for `amount` is wrong; the withdrawable amount is capped at the contract's aToken balance, not the initiator's max withdrawable amount.

Morpho: All issues have been fixed in the following commits:

1. 0d9a491c.
2. ea87bf7d.
3. e9801c26.
4. 4ab807f5.
5. 4ab807f5.
6. 93ea9af1.
7. 531b4d56.

Spearbit: Verified, all issues have been fixed as recommended.

5.3.2 Risks of unlimited approval of external contracts by adapters

Severity: Informational

Context: GeneralAdapter1.sol#L67, GeneralAdapter1.sol#L105, GeneralAdapter1.sol#L129, CompoundV2MigrationAdapter.sol#L50, CompoundV3MigrationAdapter.sol#L36

Description: Some adapters may approve arbitrary external contracts with unlimited approval of ERC-20 tokens, for example:

- `GeneralAdapter1.erc20WrapperDepositFor()`.
- `GeneralAdapter1.erc4626Mint()`.
- `GeneralAdapter1.erc4626Deposit()`.
- `CompoundV2MigrationAdapter.compoundV2RepayErc20()`.
- `CompoundV3MigrationAdapter.compoundV3Repay()`.

Adapters may temporarily hold user funds during a bundle execution. During the execution, if an attacker contract receives a callback, it could potentially exploit the existing approval to itself, and transfer tokens from the adapter to an arbitrary address.

However, this attack does assume that the attacker contract receives a call, either directly or indirectly. Some examples of receiving the call indirectly can be (1) transferring native tokens to the contract or (2) swapping through a malicious ERC-20 token or pool (as an intermediate step) during the swap.

It can be a trust assumption that the users should be fully aware of the external calls in the bundle they execute. In this case, users are assumed not to interact with the attacker contract or malicious tokens in the first place.

Recommendation: As a precautionary method, consider clearing the approval after each action or approving only the exact amount if applicable. This will reduce the risks of unlimited approval and attack vectors, especially if future integrations with external protocols or new adapters can be added.

Morpho: Fixed in PR 194. Approvals are now only temporary when the approved contract is either user-provided or upgradeable.

Spearbit: A problem with the `approveMaxTo` → `call` → `approveZeroTo` pattern is there are strange tokens that revert when `approve()` is called with zero allowance, such as BNB on mainnet. As a result, all functions that explicitly clear approval at the end of execution will revert when called with BNB.

Morpho: Acknowledged. This an acceptable edge case and we will make a specialized adapter if needed.

5.3.3 `EthereumGeneralAdapter1.stakeEth()` is wrongly marked as payable

Severity: Informational

Context: EthereumGeneralAdapter1.sol#L91-L95

Description: According to the discussion in PR 103, for adapter functions that use native ETH, the Morpho team standardized that users should transfer ETH into the adapter before calling the function (as opposed to calling the function with value). This is consistent with adapter functions that use ERC-20 tokens. However, `EthereumGeneralAdapter1.stakeEth()` is marked as `payable`:

```
function stakeEth(uint256 amount, uint256 maxSharePriceE27, address referral, address receiver)
    external
    payable
    onlyBundler
{
```

This makes it possible to call `stakeEth()` with value, which contradicts the discussion in the PR above.

Recommendation: Remove `payable` from `stakeEth()`.

Morpho: Fixed in commit 4e604d12.

Spearbit: Verified, the recommended fix was implemented.