

Maple Finance Security Review

Auditors

0xLeastwood, Lead Security Researcher

Alireza Arjmand, Lead Security Researcher

MiloTruck, Lead Security Researcher

Kankodu, Security Researcher

Report prepared by: Lucas Goiriz

November 25, 2025

Contents

1 About Spearbit **2**

2 Introduction **2**

3 Risk classification **2**
3.1 Impact . . . 2
3.2 Likelihood . . . 2
3.3 Action required for severity levels . . . 2

4 Executive Summary **3**

5 Findings **4**
5.1 High Risk . . . 4
5.1.1 Redeem queue can get bricked during mass exits . . . 4
5.2 Informational . . . 4
5.2.1 Minor improvements to code and comments . . . 4
5.2.2 Implementation of `previewRedeem` could be dangerous . . . 5
5.2.3 `processRedemptions` can be DoS'd . . . 6
5.2.4 Withdrawal manager queue ordering caveat . . . 6

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

Maple is redefining asset management for the digital age — onchain, transparent, and built to scale. Through institutional-grade lending and yield strategies, Maple brings the rigor of traditional finance to the transparency and innovation of crypto. This is asset management, evolved for the future, global, onchain financial system.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Maple Finance 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 8 days in total, Maple Finance engaged with Spearbit to review withdrawal-manager-queue at commit hash d2725904 (tag v2.0.0-rc.1).The review focused on the following scope:

Withdrawal-Manager-Queue `v2.0.0-rc.1`

Path	Change Description	Mainnet Action
`contracts/MapleWithdrawalManager.sol`	Updated Withdrawal manager that allows one address to have multiple requests	Upgrade
`contracts/proxy/MapleWithdrawalManagerMigratorV200.sol`	Migrator to upgrade current withdrawal managers on mainnet to the new v200 implementation	New Deployment
`contracts/utils/SortedArray.sol`	Utility to manage users requests	New Deployment (part of MapleWithdrawalManager)

The rest of the scope will be released at a later date when the code is published

In this period of time a total of **5** issues were found.

Summary

Project Name	Maple Finance
Repository	withdrawal-manager-queue
Commit	d2725904 (tag v2.0.0-rc.1)
Type of Project	DeFi, Borrowing & Lending
Audit Timeline	Oct 6th to Oct 14th

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	0	0	0
Low Risk	0	0	0
Gas Optimizations	0	0	0
Informational	4	1	3
Total	**5**	**2**	**3**

5 Findings

5.1 High Risk

5.1.1 Redeem queue can get bricked during mass exits

Severity: High Risk

Context: *(No context files were provided by the reviewer)*

Description: When users submit a `redeemRequest` to withdraw their shares from the pool, their request ID is queued in a sorted array stored in the `MapleWithdrawalManager`. During `processRedemptions()`, this array is processed sequentially from start to finish. Once a request's shares are fully redeemed, the request is removed from the array. The removal process begins at the first index and proceeds to the last. During the removal process process, all subsequent elements are shifted one position to the left, as shown in the `SortedArray` code:

```
for (uint256 i = index_; i <= length_ - 2; i++) {
    uint256 nextValue_              = array.values[i + 1];
    array.values[i]                 = nextValue_;
    array.valueToIndex[nextValue_] = i;
}
```

This continuous rewriting of storage can consume a significant amount of gas, potentially blocking the `processRedemptions` call in the `MapleWithdrawalManager`.

Impact: This issue can:

1. Cause withdrawal requests to be blocked during a mass exit, or...
2. Be exploited by an attacker who submits many small withdrawal requests to congest the queue.

To mitigate this, admins can call `removeRequest()` on later requests in the queue to clear space and continue processing. However, calling `removeRequest()` directly can cause users interacting through a router to lose access to their funds.

Recommendation: Consider implementing a linked list to minimize storage operations when removing elements from the array.

Maple Finance: Acknowledged the issue. The linked list approach is ready for review and merged at commit 28b18e0.

Spearbit: The linked list now removes elements from the array correctly, with significantly reduced complexity and reasonable gas usage. However, another issue poses DoS risks that is yet to be resolved.

It would still possible to DoS the withdrawal manager by submitting many requests to redeem shares and then calling `removeSharesById()` on the router to zero out the `queue.requests` mapping value. Now each time `_processRequest()` is called when iterating through the queue it will essentially skip this item as it has no shares to process. But something key to note is that if the number of unprocessed requests that have been "removed" by the owner exceeds block gas limits, even if the `onlyRedeemer` role called `processRedemptions()` in the withdrawal manager of a single share, it would be effectively be impossible to clear these out.

Maple Finance: Acknowledged, on the additional issue with empty requests which will be fixed in a follow up PR.

Spearbit: Verified fix. The issue has been addressed in and the changes are merged at commit 76a8ba7. The `onlyRedeemer` role now has the ability to clear out empty redemption requests without processing non-empty requests.

5.2 Informational

5.2.1 Minor improvements to code and comments

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: There are multiple minor issues across the protocol that can be fixed to increase the quality of the code:

1. MapleWithdrawalManager.sol#L248-L253: This function can be declared `external`.
2. MapleWithdrawalManager.sol#L227: This transfer can be moved out of the for-loop to transfer `sharesToRemove_`, instead of performing multiple transfers.

Maple Finance: The issues have been addressed as follows:

1. Fixed in commit 31481c1.
2. Fixed in commit 2738521.

Spearbit: Verified fixes.

5.2.2 Implementation of `previewRedeem` could be dangerous

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: In the `MapleWithdrawalManager` contract, `previewRedeem()` is implemented as follows:

```
uint256 sharesAvailable_ = manualSharesAvailable[owner_];

if (sharesAvailable_ == 0) return ( 0, 0 );

require(shares_ <= sharesAvailable_, "WM:PR:TOO_MANY_SHARES");

( redeemableShares_, resultingAssets_ ) = _calculateRedemption(shares_);
```

However, this implementation violates EIP-4626 and makes it somewhat dangerous for an integrator to use `previewRedeem()` to estimate redemptions.

1. The function checks for `manualSharesAvailable[owner]`, which it should not according to the EIP:

> MUST NOT account for redemption limits like those returned from maxRedeem and should always act as though the redemption would be accepted, regardless if the user has enough shares, etc...

The worst case is it returns `resultingAssets = 0` if `manualSharesAvailable[owner] == 0`, so if `MaplePool.previewRedeem()` is called for an owner with no manual shares and any non-zero share amount, it returns `0`.

2. There should be a `redeemableShares_ == shares_` check. Otherwise, if `availableLiquidity_ < requiredLiquidity_` in `_calculateRedemption()`, `resultingAssets_` would be smaller than `shares_` converted to assets. This seems dangerous if any integrator expects `previewRedeem()` to return the asset value of `shares_` at the current share price.

Recommendation: Consider the following:

1. Do not check `manualSharesAvailable` in the function.
2. Add a `redeemableShares_ == shares_` check.

Maple Finance: Acknowledged. Whilst we do maintain to the best of our ability ERC-4626 compliance, due to the nature of async redemptions, redeem related functions may not 100% adhere to the standard and we would not want to update this implementation. We would advise teams that integrate to use `convertToExitAssets()` to clarify how much their position is worth; `previewRedeem()` doesn't make sense for most users/integrators to use as we will push tokens to them.

Spearbit: Acknowledged.

5.2.3 `processRedemptions` can be DoS'd

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: In `MapleWithdrawalManager`, `_processRequest()` directly calls `MaplePool.redeem()`:

```
// Otherwise, just adjust totalShares and perform the redeem.
totalShares -= processedShares_;

IPoolLike(pool).redeem(processedShares_, request_.owner, address(this));
```

Since `redeem()` transfers assets to `request_.owner`, it is possible for users to intentionally DoS redemptions if they can force the transfer to revert. For example, cbBTC has a blacklist. A malicious owner who is blacklisted can intentionally create withdrawal requests, causing `redeem()` to revert when called. As a result, `processRedemptions()` can be DoS'd.

Note that this is not applicable to pools which funds go through the router as users will never interact with `MapleWithdrawalManager`.

Recommendation: The Maple team has clarified on why this does not need to be fixed below. However, in the worst-case scenario, malicious requests can be removed via `removeRequest()`.

Maple Finance: Acknowledged, this is known from the initial set of audits for the queue-based withdrawal manager, because of the nature of the push pattern (non redeem flow) we didn't want the processing of the queue to be blocked the admin is able to kick out an address that has been blacklisted.

In the case of the pull flow (`manualSharesAvailable`) this wouldn't DoS processing of redemptions as it's on the user to call redeem themselves. But in the case of `syrupBTC` where the router is processing the redemptions and pulling assets from the Pool it was important to also add a `removeRequests()` equivalent on the router which we have so we are able to kick out blacklisted users.

Additionally in the case where a user is partially processed on the withdrawal manager we are not able to remove that user's request anymore as it's processed therefore when calling `processRedemptions()` on the router we can skip that `requestId` to not block other users.

Our withdrawal bot will also run checks to see if a wallet is blacklisted before processing the main withdrawal manager queue to reduce the likelihood of partially processing a blacklisted user (whilst still possible due to the non atomic nature of the two sets of `processRedemptions()` transactions).

In short due to the provisions above the queue should never be DoS'd.

Spearbit: Acknowledged.

5.2.4 Withdrawal manager queue ordering caveat

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: While the implementation of the `MapleWithdrawalManager` contract will be generally used by various users and contracts interacting with individual Maple pools, there are some concerns regarding redemption processing that should be considered when making any future protocol changes.

When the `onlyRedeemer` role processes redemptions in the withdrawal contract, all addresses that are marked for manual withdrawal will re-process the redemption calculation that was also done in `MapleWithdrawalManager._processRequest()`. In between this time, the shares waiting to be processed manually by the router will continue to accrue yield and can be instantly redeemed, bypassing the queue system.

Recommendation: In the case where different sets of users are interacting with the same queue system, it's important that any address configured with `isManualWithdrawal` also enforces the queue order when manually processing the exit. This seems to get complex when considering ordering issues between various enabled router contracts.

Ensure `isManualWithdrawal` is not enabled for complex queue ordering between multiple addresses.

Maple Finance: Acknowledged.

Spearbit: The issue includes recommendations for future actions and there is no need for a fix.