macro

Maple Finance A-1

Security Audit

December 15, 2023
Version 2.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for Maple Finance's smart contract code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team from November 27, 2023 to December 11, 2023.

The purpose of this audit is to review the source code of certain Maple Finance Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro’s review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

Severity	Count	Acknowledged	Won't Do	Addressed
Medium	2	1	-	1
Low	1	-	-	1
Code Quality	12	6	2	4
Informational	3	-	-	-

Maple Finance was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions with the Maple Finance team.
- Available public documentation and provided docs for the specific release.

Source Code

The following source code was reviewed during the audit:

- **Repository:** maple-core-v2
- **Commit Hash:** `a2cf31c51e3efa44ae17ef5ab4bfc1ea2581c112`

Specifically, we audited the following contracts.

Source Code	SHA256
modules/globals/contracts/MapleGlobals.sol	`449d783d0e35f05eb0632dcb5f5a4c84ed0178de79fd5878ddb40dcd1754f5a4`
modules/fixed-term-loan/contracts/MapleLoan.sol	`9e41c80e34c9654f9c36b73d64019e14310dc5b8cd8dc9056447cde37a79c814`
modules/fixed-term-loan/contracts/MapleLoanFactory.sol	`e9958312abff174b3a66d88d75d1eb2fdae8d1764fd6ec8d4d10ec2e9e03922c`
modules/fixed-term-loan/contracts/MapleLoanV502Migrator.sol	`6aeb640d1596775b3d20df42b7418fb74cdcb850b8c1e6ec4774bcb2f5a1ad0b`
modules/withdrawal-manager-cyclical/contracts/MapleWithdrawalManager.sol	`7a9429ae3022113e27250216a6a73ff85de66ca1bb18979393ce2ecd1854ea25`
modules/withdrawal-manager-cyclical/contracts/MapleWithdrawalManagerInitializer.sol	`40692debf738632d8fc67cb3ee4f2fcfd21477a63e38715b7fb45d4728153ac6`
modules/pool/contracts/MaplePoolManager.sol	`917e6bab44c1d62202c62a23932fd31cad8c75a441bec9b598b67a87c3629f60`
modules/pool/contracts/MaplePoolDeployer.sol	`d3ed4259a4fe93c1d3a0c30dfc782f51767164fd4c4ce86da2893e88079cde0b`

Source Code	SHA256
modules/pool/contracts/proxy/MaplePoolManagerInitializer.sol	bc2ce955a74c77841b6518a1e3edf8401f4eb3aee2d987e52d45bb220122c970
modules/pool/contracts/proxy/MaplePoolManagerMigrator.sol	515ac543af75401de5b2eb892dba497c87d2772b2a53994e97d1ab704934f8b3
modules/pool/contracts/proxy/MaplePoolManagerStorage.sol	9a9b420c5e9d1945f4a357a4040c00085cd210db44e10cd4a4a486f1e6130123
modules/pool/contracts/proxy/MaplePoolManagerWMMigrator.sol	80c550a681809f19e55563dda05c33bc61124ba7d1d56440d298166d434db461
modules/pool-permission-manager/contracts/MaplePoolPermissionManager.sol	6a20e9a8a59fe6e7f6d0698bc99e5d8db00132360ddfd29693feb462e6ddf095
modules/pool-permission-manager/contracts/proxy/MaplePoolPermissionManagerInitializer.sol	48e1bc5cbf2e5bbac9bb46ce120d99219833d4c578394a2b72b2cb27127c1d29
modules/pool-permission-manager/contracts/proxy/MaplePoolPermissionManagerStorage.sol	f6170f0d8e0f749c7342347b35474295ae199ce504992e896cf69796cae65bb0
modules/withdrawal-manager-queue/contracts/proxy/MapleWithdrawalManagerFactory.sol	553db6caa8fe3ed727e53aecdc204e37f589241c6e840304cca34f7fe7a1acfa
modules/withdrawal-manager-queue/contracts/proxy/MapleWithdrawalManagerInitializer.sol	48ff3f48b01ff3b9b40f80b55b9f817b4715452a456cf0658507a16ee6d0150f
modules/withdrawal-manager-queue/contracts/proxy/MapleWithdrawalManagerStorage.sol	d24307f9400c8208247915c0753b06b29c34bb43911b8ad3c893f95be6630720
modules/withdrawal-manager-queue/contracts/MapleWithdrawalManager.sol	68b73210a1b7d82629f9f5ca28b9fabb7e383616329140acd86181720d08e1c5

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

Security Level Reference

We quantify issues in three parts:

1. The high/medium/low/spec-breaking **impact** of the issue:
    - How bad things can get (for a vulnerability)
    - The significance of an improvement (for a code quality issue)
    - The amount of gas saved (for a gas optimization)
2. The high/medium/low **likelihood** of the issue:
    - How likely is the issue to occur (for a vulnerability)
3. The overall critical/high/medium/low **severity** of the issue.

This third part – the severity level – is a summary of how much consideration the client should give to fixing the issue. We assign severity according to the table of guidelines below:

Severity	Description
(C-x) Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant funds/assets WILL be lost.**
(H-x) High	We recommend the client **must** address the issue, no matter what, because not fixing would be very bad, *or* some funds/assets will be lost, *or* the code's behavior is against the provided spec.
(M-x) Medium	We recommend the client to **seriously consider** fixing the issue, as the implications of not fixing the issue are severe enough to impact the project significantly, albiet not in an existential manner.
(L-x) Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
(Q-x) Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain composability, developer ergonomics, or even certain aspects of protocol design.
(I-x) Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
(G-x) Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

M-1 Actors in manual redeem mode may remain in processed but not redeemed state for a long time

TOPIC	STATUS	IMPACT	LIKELIHOOD
Protocol Design	Acknowledged	Medium	Low

Actors for which manual redeem mode is enabled may go against the purpose of the withdrawal queue mechanism by performing step 1 of the redeem flow (getting their request processed) but delaying step 2, exchanging shares for the available assets indefinitely until an appropriate time.

As a result, they may have an advantage in certain market conditions when an unrealized loss is about to increase, which they may front-run as they can perform exchange instantly while others would need to go through the withdrawal manager queue.

RESPONSE BY MAPLE FINANCE

> We will consider this for the version 2 of the withdrawal mechanism design.
>
> Currently, we do not believe this to be a concern, as the manual flow can only be enabled for LPs by the Pool Delegate or Protocol Admins. This will be done only for over-collateralized pools, such as our cash management product where the incentive for parking funds is lower.
>
> This flow is also designed for smart contract integrators, so we will ensure that we review the smart contracts ahead of time to see if this would be an issue. As an added precaution, we will ask our Pool Delegates and Protocol Admins to use private RPCs like Flashbots to send any transactions that would reduce the share token exchange rate, to further discourage this issue.
>
> If an LP is seen abusing this, we can also revoke their permissions to interact with the share token or not process any future redemption requests.

~~M-2~~ Maple Withdrawal Manager may be created with values that break invariant

TOPIC	STATUS	IMPACT	LIKELIHOOD
Input Validation	Fixed	Medium	Low

In the withdrawal-manager-cyclical module, `MapleWithdrawalManagerInitializer._initialize()` does not perform safe downcasting of provided inputs, such as `startTime_`, `cycleDuration_`, and `windowDuration_`. As a result, the provided system invariant may be broken.

> The system is designed to reject any attempts to set a start time that falls in the past.

If provided `startTime_`, `cycleDuration_`, and `windowDuration_` are greater than uint64, their values will pass initial checks such as `startTime_ >= block.timestamp`, but these values will get truncated on assignment, and incorrect values will be set in cycleConfigs[0]. Allowing even startTime to be 0.

```
function _initialize(address pool_, uint256 startTime_, uint256 cycleDuratic
        require(pool_           != address(0),      "WMI:ZERO_POOL");
        require(startTime_      >= block.timestamp, "WMI:INVALID_START");
        require(windowDuration_ != 0,               "WMI:ZERO_WINDOW");
        require(windowDuration_ <= cycleDuration_,  "WMI:WINDOW_OOB");

        pool        = pool_;
        poolManager = IPoolLike(pool_).manager();

        cycleConfigs[0] = CycleConfig({
            initialCycleId:   1,
            initialCycleTime: uint64(startTime_),
            cycleDuration:    uint64(cycleDuration_),
            windowDuration:   uint64(windowDuration_)
        });

        emit Initialized(pool_, cycleDuration_, windowDuration_);
        emit ConfigurationUpdated({
            configId_:         0,
            initialCycleId_:   1,
```

```
            initialCycleTime_: uint64(startTime_),
            cycleDuration_:    uint64(cycleDuration_),
            windowDuration_:   uint64(windowDuration_)
        });
    }
```

Proof of concept:

```
function test_createInstance_success() external {
   // @audit - startTime set to large value
   bytes memory calldata_ = abi.encode(address(pool), uint256(type(uint64).m

    MapleWithdrawalManager withdrawalManager_ = MapleWithdrawalManager(factc

    (
        uint64 initialCycleId_,
        uint64 initialCycleTime_,
        uint64 cycleDuration_,
        uint64 windowDuration_
    ) = withdrawalManager_.cycleConfigs(0);

    assertEq(withdrawalManager_.pool(),           address(pool));
    assertEq(withdrawalManager_.latestConfigId(), 0);

    assertEq(initialCycleId_,   1);
    // @audit - initialCycleTime should never be smaller than block.timestam
    // but because of truncation of values larger than type(uint64).max it e
    assertEq(initialCycleTime_, 0);
    assertEq(cycleDuration_,    7 days);
    assertEq(windowDuration_,   2 days);
}
```

Remediations to consider:

- Update implementation to perform safe downcasting and revert in case of out-of-range values.

~~L-1~~ `hasPermissions()` will return true if there are no lenders passed

TOPIC	STATUS	IMPACT	LIKELIHOOD

Access control	Fixed	Medium	Low

`MaplePoolPermissionManager`'s `hasPermission` function accepts an arbitrary length array for `lenders_` and loops through all addresses passed to check the corresponding pool permission level and bitmap. However, if no lenders are passed to this function, the returned value will be `true`, potentially allowing access to gated functions without permission.

It is worth mentioning that this is not possible in the current codebase, as it's impossible to provide arbitrary values to this function outside of external off-chain calls to the view function.

Consider returning `false` if no lenders are provided. Alternatively, document this behavior to avoid pitfalls.

~~Q-1~~ MaplePoolPermissionManager's onlyGovernor modifier is unused

TOPIC	STATUS	QUALITY IMPACT
Unnecessary code	Fixed	Low

Consider removing `onlyGovernor` modifier from the `MaplePoolPermissionManager` contract since it is not used.

~~Q-2~~ Missing indexed attributes for multiple events

TOPIC	STATUS	QUALITY IMPACT
Events	Fixed	Low

For the following events, consider using an indexed attribute for one or more parameters:

- IMaplePoolPermissionManager
  - LenderAllowlistSet
  - PermissionAdminSet
  - PoolBitmapsSet
  - PoolPermissionLevelSet
- IMapleWithdrawalManagerInitializer
  - Initialized
- IMapleWithdrawalManager
  - RequestCreated (for the owner parameter)

RESPONSE BY MAPLE FINANCE

> Additional changes in https://github.com/maple-labs/withdrawal-manager-queue/commit/9822b7551a11aab5f97be761df34d08f69973fdd.

Q-3 Redundant assignments in the MapleWithdrawalManagerFactory

TOPIC	STATUS	QUALITY IMPACT
Unnecessary code	Wont Do	Low

In both withdrawal-manager-cyclical and withdrawal-manager-queue modules, the implementation of `MapleWithdrawalManagerFactory.createInstance()` function contains redundant and unnecessary assignment.

```
isInstance[instance_ = super.createInstance(arguments_, salt_)] = true;
```

Consider removing the mentioned assignment since `super.createInstance()` performs the same in the parent function implementation.

```
isInstance[instance_] = true;
```

RESPONSE BY MAPLE FINANCE

> We won't be redeploying the cyclical withdrawal manager factory. For ease of maintainability we'll keep the same assignment in the new queue withdrawal manager factory.

Q-4 Inconsistent whenProtocolNotPaused behavior between Withdrawal Managers

TOPIC	STATUS	QUALITY IMPACT
Access control	Acknowledged	Low

Across the two withdrawal manager contract types (queue and cyclical) there are logic inconsistencies in implementing pausable functionality.

The `whenProtocolNotPaused` modifier checks for `isFunctionPaused` in the queue withdrawal manager.

```
modifier whenProtocolNotPaused() {
    require(!IGlobalsLike(globals()).isFunctionPaused(msg.sig), "WM:PAUSED")
    _;
}
```

However, the `whenProtocolNotPaused` modifier in the cyclical manager checks `protocolPaused` which is less flexible and does not allow fine-grained control.

```
modifier whenProtocolNotPaused() {
    require(!IGlobalsLike(globals()).protocolPaused(), "WM:PROTOCOL_PAUSED")
    _;
}
```

Consider updating implementation to provide consistent behavior between Withdrawal Managers of different types to reduce maintenance impact and prevent operational issues.

RESPONSE BY MAPLE FINANCE

> Currently the protocol pause is only used for one function setExitCycleConfig() in the cyclical withdrawal manager. We will consider update in the future release.

~~Q-5~~ Improper condition in _processManualExit()

TOPIC	STATUS	QUALITY IMPACT
Best practices	Fixed	Low

In the `MapleWithdrawalManager`, `_processManualExit()` has the following implementation.

```
function _processManualExit( uint256 shares_, address owner_)
    internal returns ( uint256 redeemableShares_, uint256 resultingAssets_)
{
    require(shares_ > 0,                                  "WM:PE:NO_SHARES");
    require(shares_ <= manualSharesAvailable[owner_], "WM:PE:TOO_MANY_SHARES

    ( redeemableShares_ , resultingAssets_ ) = _calculateRedemption(shares_)

    // @audit - case when shares_ < redeemableShares_ is not reachable
    require(shares_ <= redeemableShares_, "WM:PE:NOT_ENOUGH_LIQUIDITY");
    ...
}
```

The value of `redeemableShares_` returned from the `_calculateRedemption()` is always represented by the following relationship `redeemableShares_ <= shares_`. Therefore, `shares_` cannot ever be smaller than `redeemableShares_`.

The same logic applies to the similar guard condition in the `processRedemptions()` function.

```
require(maxSharesToProcess_ <= redeemableShares_, "WM:PR:LOW_LIQUIDITY");
```

Consider changing conditions in the implementation above to the following to indicate that only full redemption would be possible.

In `_processManualExit()`

```
require(shares_ == redeemableShares_, "WM:PE:NOT_ENOUGH_LIQUIDITY");
```

and in `processRedemptions()`

```
require(maxSharesToProcess_ == redeemableShares_, "WM:PR:LOW_LIQUIDITY");
```

~~Q-6~~ Inconsistent access control for upgrading withdrawal managers

TOPIC	STATUS	QUALITY IMPACT
Access control	Fixed	Low

Maple Finance now features two types of withdrawal managers: Cyclical and Queue withdrawal managers. Both of these withdrawal managers feature upgrade functionality, which is access-controlled. However, access control is inconsistent.

Upgrading the cyclical withdrawal manager is allowed for the pool delegate **or governor**, while in the case of the queue withdrawal manager, upgrades are allowed to be performed by the pool delegate **or security admin**.

```
require(msg.sender == poolDelegate_ || msg.sender == governor(), "WM:U:NOT_A
```

```
require(msg.sender == poolDelegate_ || msg.sender == securityAdmin(), "WM:U:
```

Consider updating access control in the Cyclical withdrawal manager to allow `securityAdmin()` instead of the `governor()`.

RESPONSE BY MAPLE FINANCE

> Fixed to allow security admin to upgrade.

Q-7 Inconsistent pause control for upgrades

TOPIC	STATUS	QUALITY IMPACT
Access control	Acknowledged	Low

For the `withdrawal-manager-cyclical` module, `MapleWithdrawalManager` proxy functions do not feature pausable behavior since `whenProtocolNotPaused` modifier is missing. This contrasts the implementation of other similar contracts across modules in the Maple Finance codebase.

Due to the above, fine-grained control over paused functionality on a specific contract instance is unavailable. However, a fallback is a pauseable functionality on the factory level. Functions missing `whenProtocolNotPaused` modifier:

- MapleWithdrawalManager.migrate()
- MapleWithdrawalManager.setImplementation()
- MapleWithdrawalManager.upgrade()

The same functions have the corresponding modifier in withdrawal-manager-queue: `MapleWithdrawalManager` .

RESPONSE BY MAPLE FINANCE

> We will make this update when introducing function level granularity for pausing in this contract in a future iteration.

Q-8 Unnecessary event emission in cyclical MapleWithdrawalManager

TOPIC	STATUS	QUALITY IMPACT
Events	Wont Do	Low

`MapleWithdrawalManager.processExit()` emits `WithdrawalCanceled` event in addition to `WithdrawalProcessed` when all lockedShares are processed, which is slightly misleading and could potentially cause errors in off-chain tracking and monitoring tools.

Consider not emitting this event when all `lockedShares` are processed.

RESPONSE BY MAPLE FINANCE

> This is a requirement from our team to help with the subgraph data modeling.

Q-9 Inconsistent access control for MapleLoan.skim()

TOPIC	STATUS	QUALITY IMPACT
Access control	Acknowledged	Low

In the `fixed-term-loan` module `MapleLoan.skim()` is accessible to anyone, while in the `open-term-loan` module `MapleLoan.skim()` is accessible only to the borrower and governor.

Consider updating access control for the `MapleLoan.skim()` in the `fixed-term-loan` module to prevent external parties from profiting from the mistakes of the protocol users.

RESPONSE BY MAPLE FINANCE

> We don't see this as an issue as there are inherit differences between open term and fixed term loans. For one, the open term loans don't directly hold funds to be drawn down like the fixed term loan. We don't expect the same behaviour across both loan types but this is something we will keep in mind in future iterations.

Q-10 Inconsistent validation in MapleLoan.skim()

TOPIC	STATUS	QUALITY IMPACT
Input Validation	Acknowledged	Low

In the `fixed-term-loan` module `MapleLoan.skim()` has no zero address check for the destination address, while in the `open-term-loan` module `MapleLoan.skim()` corresponding check is present.

Consider adding a zero address check to the `MapleLoan.skim()` in the `fixed-term-loan` module to prevent assets from being inadvertently locked.

RESPONSE BY MAPLE FINANCE

> As we are not updating the fixed term loan in this release we will defer this to a future update.

Q-11 Unnecessary import in MapleRefinancer

TOPIC	STATUS	QUALITY IMPACT
Unnecessary code	Acknowledged	Low

In the open-term-loan module, `MapleRefinancer` contains unused import of IERC20.

Consider removing this unnecessary import.

RESPONSE BY MAPLE FINANCE

> We will update the MapleRefinancer in a future release when we decide to re-deploy the MapleRefinancer contract.

Q-12 Use onlyPoolManager modifier in the cyclical withdrawal manager

TOPIC	STATUS	QUALITY IMPACT
Best practices	Acknowledged	Low

In the withdrawal-manager-cyclical module, the `MapleWithdrawalManager` contract implements pool manager access control directly in functions `addShares()`, `removeShares()`, and `processExit()`.

```
require(msg.sender == poolManager, "WM:AS:NOT_POOL_MANAGER");
```

However, in the Queue withdrawal manager this access control is implemented using `onlyPoolManager()` modifier.

Consider replacing repetitive access controls with the `onlyPoolManager()` modifier in the cyclical withdrawal manager.

RESPONSE BY MAPLE FINANCE

> We will update in a future release when we update the cyclical withdrawal manager.

I-1 Default bitmaps for Pool and Function level permission allow permissionless access

TOPIC
Access control

IMPACT
Informational ✳

In the pool-permission-manager module, the `MaplePoolPermissionManager` contract implements several modes for access control. Two modes that allow function level control and higher level but less fine-grained pool level control are represented by `FUNCTION_LEVEL` and `POOL_LEVEL` constant values.

These modes require corresponding bitmaps to be set in `poolBitmaps` mapping storage variable for each pool. When values in specific bitmap match bitmaps of lenders from respective `lenderBitmaps` mapping entries, access is allowed; otherwise is denied.

However, **the default when corresponding bitmaps in these two modes are not set is to allow anyone access**. This default comes with a high risk of misconfiguration, and actors responsible for configuring the permission manager (pool delegate and protocol admins) must take necessary precautions.

While this design choice is intentional, and it is meant to allow transfers to be allowed by default, Maple Finance will need to ensure that pool delegates are aware of misconfiguration risks.

RESPONSE BY MAPLE FINANCE

> This is already in the natspec to ensure no operator error. However, we will highlight this in the docs too.

I-2 Config change delay period may be too short

TOPIC	IMPACT
Protocol Design	Informational ✳

In the cyclical withdrawal manager, the delay period when the new configuration starts to apply is +3 from the current cycle. This allows all users who have already locked their shares not to be affected during withdrawal (in case of enough liquidity during withdrawal).

```
When the pool delegate changes the configuration, it will take effect only c
This way all users that have already locked their shares will not have their

    C1       C2       C3             C4
|===--.--|=====-----|=====-----|==========----------|
      ^                    ^
configuration change     new configuration kicks in

Users that request a withdrawal during C1 will withdraw during WW3 using the
Users that lock their shares during and after C2 will withdraw in windows th
```

However, if a delay is meant to be used as a Timelock mechanism to allow those unhappy with the new configuration to exit, the defined delay may be too short. Depending at which moment the new config is set in the current cycle, those users who haven't locked their shares yet may have time from `0 < time-to-decide < cycleDuration` to decide on their withdrawal. This, in extreme cases, is a too short time period for the user to react.

Consider increasing the delay to +4, so that users who haven't yet locked their shares have at least one full cycle to decide on their withdrawal.

RESPONSE BY MAPLE FINANCE

> We don't see this as an issue as the user will still be able to request to withdraw in C2 and be able to exit in C4 as the window duration will still be at the start of C4. But we will keep this in mind for future designs.

I-3 Not checking for L2 sequencer in Base contracts

TOPIC
Oracles

IMPACT
Informational ✳

As suggested by Chainlink documentation, Optimistic L2 oracle implementations should check their corresponding L2 Sequencer uptime feed to ensure that the sequencer is live before trusting the data feed returned by the oracle. Even with the Oracle staleness price check, if the sequencer is down, there could be multiple price changes provided by the Oracle that won't go through and will get queued and applied once the sequencer is up; if price changes are high, it could cause unfair conditions for users and borrowers.

Maple **Base** Oracles are currently not being used by the protocol; if enabled checks should be added to verify that the sequencer is live and provide a grace period to react to potential high price changes (See suggested example in Chainlink docs).

RESPONSE BY MAPLE FINANCE

> This is something we will keep in mind if we decide to use oracles on the BASE/L2 deployments

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Maple Finance team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.