Three Sigma Labs

Code Audit

MAPLE

MAPLE Lending Protocol

Disclaimer

Code Audit

MAPLE Lending Protocol

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

MAPLE Lending Protocol

Table of Contents

Summary

Code Audit

MAPLE Lending Protocol

Summary

Three Sigma Labs audited Maple Finance's V2 smart contracts Q4 update in a 4 person week engagement. The audit was conducted from 6-11-2023 to 17-11-2023.

Protocol Description

Maple Finance is an institutional crypto-capital network, which provides the infrastructure for credit experts to efficiently manage and scale crypto lending businesses and connect capital from institutional and individual lenders to innovative, blue-chip companies.

This new release of the Maple V2 smart contracts introduced the following components:

- New FIFO queue based withdrawal manager submodule to be used with permissioned pools.
- Pool Permission Manager submodule which will govern if LPs have permission to interact with Maple Pools.
- Redeployment of the Fixed Term Loan Factory to include access controls.
- New protocol actor called the Operational Admin
- Minor update to the current cyclical withdrawal manager to allow a start time for the first cycle to be set.

Scope

Code Audit

MAPLE Lending Protocol

Scope

```
Globals v2 / contracts
└── MapleGlobals.sol

Fixed Term Loan / contracts
├── MapleLoan.sol
├── MapleLoanFactory.sol
└── MapleLoanV502Migrator.sol

Withdrawal Manager Cyclical / contracts
├── MapleWithdrawalManager.sol
└── MapleWithdrawalManagerInitializer.sol

Pool v2 / contracts
├── MaplePoolManager.sol
├── MaplePoolDeployer.sol
└── proxy/
    ├── MaplePoolManagerInitializer.sol
    ├── MaplePoolManagerMigrator.sol
    ├── MaplePoolManagerStorage.sol
    └── MaplePoolManagerWMMigrator.sol

Pool Permission Manager / contracts
├── MaplePoolPermissionManager.sol
└── proxy/
     ├── MaplePoolPermissionManagerInitializer.sol
     └── MapleWPoolPermissionManagerStorage.sol

Withdrawal Manager Queue / contracts
├── MapleWithdrawalManager.sol
└── proxy/
    ├── MapleWithdrawalManagerFactory.sol
    ├── MapleWithdrawalManagerInitializer.sol
    └── MapleWithdrawalManagerStorage.sol
```

Assumptions

The scope of the audit was carefully defined to include the contracts at the lowest level of the inheritance hierarchy, as these are the ones that will be deployed to the mainnet. The libraries used in the implementation of these contracts are internal contracts that have been previously audited and shown to be secure.

Methodology

Code Audit

MAPLE Lending Protocol

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

MAPLE Lending Protocol

Project Dashboard

Application Summary

Name	Maple Finance
Commit	c707041b
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	6 to 17 November, 2023
№ of Auditors	2
Review Time	4 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	3	2	1

None	4	3	1

Category Breakdown

Suggestion	3
Documentation	1
Bug	2
Optimization	1
Good Code Practices	0

Code Maturity Evaluation

Code Audit

MAPLE Lending Protocol

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 as well as takes the correct measures in rounding the results of arithmetic operations.
Centralization	Weak. The introduced Operational Admin may control the pool delegates.
Code Stability	Satisfactory. The code was stable during the audit.
Upgradeability	Moderate. Certain smart contract implementations can be modified after deployment, albeit with proper timelocks and functional upgradeability patterns.
Function Composition	Satisfactory. Certain components are similar, and the codebase would benefit from increased code reuse.
Front-Running	Moderate. Processing redemptions could be frontrunned and revert [3S-MAPLE-L01].
Monitoring	Satisfactory. Events are correctly emitted and Maple Finance keeps track of on chain activity.
Specification	Satisfactory. In-depth and well structured high-level specification as well as codebase documentation.
Testing and Verification	Satisfactory. Extensive test code coverage as well as usage of tools and different test methods.

Findings

Code Audit

MAPLE Lending Protocol

Findings

3S-MAPLE-L01

Queue MapleWithdrawalManager may revert due to honest removeShares() or manual redeem calls

Id	3S-MAPLE-L01
Classification	Low
Category	Bug
Status	Addressed in #7dbcd37

Description

The queue withdrawal manager limits the **`sharesToProcess`** in **`processRedemptions()`** to the **`totalShares`**.

This might make the call revert if non malicious users remove shares or do manual redeems.

Another concurrency issue is if the available liquidity is just enough to cover for a **`processRedemptions()`** call, but someone frontruns it and does a manual redeem.

Recommendation

Instead of reverting if the **`processedShares`** argument is bigger than **`totalShares`**, limit it:

```
function processRedemptions(uint256 sharesToProcess_) external override
whenProtocolNotPaused nonReentrant onlyRedeemer {
    ...
    uint256 cachedTotalShares_ = totalShares;
    if (sharesToProcess_ > cachedTotalShares_) sharesToProcess_ =
cachedTotalShares_;
    ...
}
```

It's trickier to solve the lack of liquidity concurrency issue, as it would require allowing partial redemptions.

3S-MAPLE-L02

Unbounded loops in **`MapleWithdrawalManager`** may break in the future if the withdrawal windows are small

Id	3S-MAPLE-L02
Classification	Low
Category	Bug
Status	Acknowledged

Description

Loops revert due to OOG errors if too many iterations are performed. Some functions will stop working if these loops grow too large.

For example, `getConfigAtId(uint256 cycleId_)` may be called with a **`cycleId_`** well in the past, leading to OOG.

The likelihood of this happening depends on the number of times the configs were changed and the cycle durations.

Due to this, it might make it impossible to call **`removeShares()`**, so the shares would be stuck.

Recommendation

Send hints in these functions (or make them internal and create 2 external functions, one with hints and another the same as the current one). For example, **`getConfigAtId(uint256 cycleId_)`** could receive **`latestConfigIdHint_`** as argument.

3S-MAPLE-L03

Factory update for maple loans can be performed with the old factory

Id	3S-MAPLE-L03
Classification	Low
Category	Suggestion
Status	Addressed in #72bc617

Description

In the MapleLoanV502Migrator, the only check is that the factory address is a valid instance.

Thus, it's possible for a loan with the old factory to be updated and use the same old factory.

Recommendation

Prevent stale updates or hardcode the new factory address (via immutable variable in the constructor).

3S-MAPLE-N01

Allow lenders to set a minimum amount of asset they would take for their shares

Id	3S-MAPLE-N01
Classification	None
Category	Suggestion
Status	Acknowledged

Description

In the queue-based version of the withdrawal manager, lenders don't control when they leave the protocol, so they don't choose when the conversion from shares to asset is performed, which may lead lenders to receive much less asset than they anticipated, since the conversion rate might have decreased substantially while they were waiting for the redeemer to process their redemption.

A particularly detrimental scenario for lenders is if they ask to redeem their shares, and then the pool delegate decides to impair a loan, followed by a call to process redemptions.

In this scenario, the loan impairment would result in a steep increase of the unrealized losses, resulting in the lenders getting much less assets than they were expecting when they asked to redeem their shares, and all of this was outside the lenders' control.

Note: if a lender chooses to perform a manual exit they can circumvent this problem.

Recommendation

Allow lenders to set a minimum amount of assets they would be willing to take for their shares (in a way similar to how exchange protocols control for slippage).

3S-MAPLE-N02

Gas savings

Id	3S-MAPLE-N02
Classification	None
Category	Optimization
Status	Addressed in #cff9348

Description

Queue **MapleWithdrawalManager**:

In _processRequest(), **`queue.requests[requestId_].shares -= processedShares_;`** can be replaced by **`queue.requests[requestId_].shares = request_.shares - processedShares_;`** to avoid reading 1 storage slot. Additionally, the line can be moved to the next else statement, to avoid writing 0 in the slot twice (subtracting and ending up with 0 and then deleting). Similarly to what is done in _removeShares().

The subtraction on line 168 of the MapleWithdrawalManager should only be performed if **`lockedShares != 0`**. This would prevent a storage read and write when **`lockedShares == 0`**, which seems the most likely scenario when calling **`addShares()`**. Recommendation: **`if (lockedShares_ != 0){ totalCycleShares[exitCycleId_] -= lockedShares_;}`**

Still on the cyclical withdrawal manager, line 250, the partialLiquidity check will always return true, since if **`(new)lockedShares_ != 0`** → **`(old)lockedShares_ != redeemableShares_`** → **`partialLiquidity`**, this check can therefore be safely removed.

3S-MAPLE-N03

Operational Admin Suggestions

Id	3S-MAPLE-N03
Classification	None
Category	Suggestion
Status	Addressed in #4be9021

Description

- **`MapleWithdrawalManager:setExitConfig()`** could be performed by the operational admin, since it can call `setPendingDelegate()` and acquire the permissions anyway.

- Delegates could choose whether they wanted the operational admin to have their permissions, basically delegating their role.

3S-MAPLE-N04

Documentation inconsistencies

Id	3S-MAPLE-N04
Classification	None
Category	Documentation
Status	Addressed in #a483c8e

Description

- The owner of **`redeem()`**, **`withdraw()`**, **`removeShares()`**, **`requestRedeem()`** and **`requestWithdraw()`** must be allowed and **`transfer()`**, **`transferFrom()`** check the sender of the shares now.

- **`setPendingPoolDelegate()`** in **`PoolManager`** can be performed by the operational admin, but is not in the docs.

- Partial redemptions are not allowed due to lack of liquidity in the queue **`MapleWithdrawalManager`** docs.

- IMapleWithdrawalManager.sol - line 79 "NOTE: The **`shares`** value is ignored.". How it is ignored ? It is used to compute the resultingAssets.