Three Sigma Labs

Code Audit

MAPLE

MAPLE Lending Protocol

Disclaimer

Code Audit

MAPLE Lending Protocol

Labs

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

Labs

Table of Contents

Disclaimer **2**
Table of Contents **3**
Summary **8**
Scope **10**
Project Dashboard **15**
Code Maturity Evaluation **18**
Code Maturity Evaluation Guidelines 18
Code Maturity Evaluation Results 19
Findings **23**
3S-MAPLE-01 23
3S-MAPLE-02 25
3S-MAPLE-03 26
3S-MAPLE-04 27
3S-MAPLE-05 29
3S-MAPLE-06 31
3S-MAPLE-07 33
3S-MAPLE-08 34
3S-MAPLE-09 35
3S-MAPLE-10 36
3S-MAPLE-11 37
3S-MAPLE-12 38
3S-MAPLE-13 39
3S-MAPLE-14 40
3S-MAPLE-15 41
3S-MAPLE-16 42
3S-MAPLE-17 43
3S-MAPLE-18 44

Summary

Code Audit

MAPLE Lending Protocol

Labs

Summary

Three Sigma Labs audited Maple Finance's V2 smart contracts in a 5 person week engagement. The audit was conducted from 24 10 2022 to 09 11 2022.

Protocol Description

Maple Finance is a decentralized credit market that uses blockchain technology to allow institutional borrowers to obtain undercollateralized loans from lending pools managed by accredited delegates. This platform enables borrowers to access loans that they may not have been able to obtain elsewhere, while also providing lenders with a way to earn sustainable returns on their assets. Delegates who create and manage pools on Maple earn a portion of the establishment fee and interest earned on the loans, and liquidity providers who add funds to the pools earn a share of the interest paid by borrowers.

Scope

Code Audit

MAPLE Lending Protocol

Labs

Scope

The audit examined Maple Finance's V2 core contracts as well as the liquidity migration contracts and procedure from V1 to V2.

V1 to V2 Protocol Liquidity Migration Contracts

maple-labs/debt-locker **(v4.0.0-rc.0**

maple-labs/loan **(v3.0.1-rc.0**

maple-labs/loan **(v3.0.2-rc.0**

maple-labs/migration-helpers (v1.0.0-rc.1

V2 Protocol Contracts

maple-labs/globals-v2 (v1.0.0-rc.0

maple-labs/liquidations (v2.0.0-rc.1

maple-labs/loan (v4.0.0-rc.1

maple-labs/maple-proxy-factory (v1.1.0-rc.0

maple-labs/pool-v2 (v1.0.0-rc.1

maple-labs/withdrawal-manager (v1.0.0-rc.1

The review was conducted on the code present in a private repository shared with Three Sigma, which contains a Foundry project with testing scripts as well as a documentation providing additional information. The code was frozen for review at commit 652d1816238e62f44cd270feb7b20ae9ca82b51f on the **maple-labs/maple-core-v2** repository which contained all the relevant modules described above.

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. No external libraries were utilized in the implementation of these contracts, so all of the relevant code was subject to review during the audit process. It is important to note that the dependencies of these contracts have been previously audited by other audit firms.

Methodology

Code Audit

MAPLE Lending Protocol

Labs

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/.

Level	Description
Critical	- Empty or freeze the contract's holdings. - Cryptographic flaws.
High	- Token holders temporarily unable to transfer holdings. - Users spoof each other. - Theft of yield. - Transient consensus failures.
Medium	- Contract consumes unbounded gas. - Block stuffing. - Griefing denial of service. - Gas griefing.
Low	- Contract fails to deliver promised returns, but doesn't lose value.
None	- Best practices. - Gas optimizations.

Project Dashboard

Code Audit

MAPLE Lending Protocol

Labs

Project Dashboard

Application Summary

Name	Maple Finance
Commit	652d1816
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	24 October to 9 November, 2022
Nº of Auditors	2
Review Time	5 person weeks

Vulnerability Summary

Issue Classification	Found
Nº Critical Severity Issues	0
Nº High Severity Issues	1
Nº Low Severity Issues	2
Nº Informational Severity Issues	Several

Category Breakdown

Suggestion	7
Optimization	7
Bug	3
Access Control	1

Code Maturity Evaluation

Code Audit

MAPLE Lending Protocol

Labs

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 governor and delegates have significant privileges over the protocol.
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.Pool delegate loan actions can be front-run, allowing users to queue up withdrawals. Front-running vulnerability when initializing shares of a pool.
Monitoring	Satisfactory. Events are correctly emitted.
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.

Automated Testing and Verification

Code Audit

Maple Finance Lending Protocol

Automated Testing and Verification

To enhance coverage of certain areas of the codebase we complement our analysis with a set of automated testing techniques:

- **Slither:** A Solidity static analysis framework with native support for multiple vulnerability detectors. We used Slither to scan the entire codebase against common vulnerabilities and programming malpractices.

Despite augmenting our security analysis, automated testing techniques still present some limitations and should therefore not be used in isolation. Slither may fail to identify vulnerabilities, either due to the lack of specific detectors or whenever certain properties fail to hold after Solidity code is compiled to EVM bytecode.

Slither Results

An initial run of Slither on Maple Finance's V2 smart contracts reported warnings considered to be false positives. Maple's codebase was equipped with automated scripting that ran Slither and the team already incorporated Slither's outputs into the development phases of the code.

Findings

Code Audit

MAPLE Lending Protocol

Labs

Findings

3S MAPLE-01

Loss of funds due to poor initialization of pool shares accounting.

Id	3S-MAPLE-01
Severity	High
Difficulty	Medium
Category	Bug

Description

When depositing into the `Pool` contract, deposited assets are converted into vault shares according to the following formula, where `totalSupply_` corresponds to the number of existing shares:

In the first case, when no shares are circulating, the number of minted vault shares is equal to the amount of assets deposited. In the second case the depositor gets minted an amount of shares proportional to its percentage of total liquidity (rounded down).

This allows for a scenario where an attacker might steal a user deposit by frontrunning the initial deposit into a vault and subsequently inflating the vault's assets.

The Pool contract has a guard that requires the number of minted shares to be larger than

zero, otherwise reverting with a `P:M:ZERO_SHARES` message. This check can be bypassed by adjusting the variables under the attacker's control (`totalSupply_` and `totalAssets()`) in order to force the rounding down for a number other than zero.

Recommendation

After deploying a new pool, require the first mint to mint a minimum number of shares that mitigates this possible attack vector.

Status

Addressed in the following PR:

https://github.com/maple-labs/pool-v2/commit/7222b846756456e33156aeddd77f4a6bc86e9b9f
https://github.com/maple-labs/globals-v2/commit/407db76e53fb0a77f8b2be78cdea49b37deb88a5

3S MAPLE-02

Inconsistent values on LoanImpaired events

Id	3S MAPLE-02
Severity	Low
Difficulty	Low
Category	Bug

Description

During the loan impairment flow, initiated by a call to the `impairLoan` function of the `PoolManager` contract, a `LoanImpaired` event is emied by both the `PoolManager.sol` and the `MapleLoan.sol` contracts. Despite having distinct signatures, both events share a `nextPaymentDueDate_` field, which is set dierently on each emission.

In the `PoolManager.sol` contract, the value is always set as the `block.timestamp`, while in a `MapleLoan` contract, it is calculated according to the following formula: `newPaymentDueDate_ = block.timestamp > originalNextPaymentDueDate_ ? originalNextPaymentDueDate_ : block.timestamp`

Recommendation

Update the variables on the events being emied to be coherent between dierent uses.

Status

Addressed by the team.

3S-MAPLE-03

The Pool Manager temporarily loses the ability to transfer from the PoolDelegate contract.

Id	3S-MAPLE-03
Severity	Low
Difficulty	High
Category	Access Controls

Description

If Pool Delegates of two `PoolManager.sol` contracts set the same address for the variable `pendingPoolDelegate` and then the new Pool Delegate calls the function `acceptPendingPoolDelegate` on both PMs, then the first accepted `PoolManager` becomes
unable to set a new Pool Delegate. It is possible to recover this functionality if the new Pool Delegate sets himself as the `pendingPoolDelegate` on the first `PoolManager` contract and then accepts it, but in doing so the functionality on the second PM is lost. Summarizing, the map variable `poolDelegates` on `GlobalsV2` contract only allows a Pool Delegate to change Pool Delegates for one `PoolManager` at a time

---

Recommendation

Update the data structure to allow a single Pool Delegate to be the Pool Delegate of multiple `PoolManager` contracts.

Status

Acknowledged by the team.

3S-MAPLE-04

Use linked list insertion hints

Id	3S-MAPLE-04
Severity	None
Difficulty	N/A
Category	Optimization

Description

The `LoanManager.sol` contract uses linked lists to store stored loan payment information according to their due dates. In order to insert a new payment in the linked list, the `_addPaymentToList(...)` function iterates over all the elements in the list. It begins iterating from the start until the due date of the payment being inserted is smaller than that of the list element. This operation has a complexity of O N , where “N” is the number of elements in the list.

The same behavior occurs in the `PoolManager` contract. The function `removeLoanManager(...)` iterates through the elements of the `loanManagerList` until it finds the correct element to be removed.

A way to improve the complexity of the search is through the use of insertion hints. Insertion hints are computed o chain and passed as calldata to the Maple protocol contracts. As an example of an insertion hint, we can consider the variables `prevId_` and `nextId_`, which give information regarding the expected previous and next nodes of the
inserted payment, upon insertion. Instead of iterating over the list to find a valid insert position, the `_addPaymentToList(...)` or the function `removeLoanManager(...)` should simply verify that the position specified by the insertion hints is valid, meaning:

- The payment at `_prevId` has a due time smaller than the one being inserted.
- The payment at `_nextId` has a due time larger than the one being inserted.
- The payment at `_prevId` points to the payment at _nextId (before insertion).

Note that this is just a simple example of an algorithm. You can also build a more eicient one using a single hint value.

Nonetheless, the algorithm has an O 1 complexity, which is far superior to the original O N complexity.

Recommendation

Modify the `_addPaymentToList(...)` and `removeLoanManager(...)` function signatures to accept insertion hints.

Status

Acknowledged by the team.

3S-MAPLE-05

`Disruption of withdrawals due to pro rata distribution of shares`

Id	3S-MAPLE-05
Severity	None
Difficulty	N/A
Category	Suggestion

Description

Users that request a withdrawal during cycle n will be able to withdraw during the withdrawal window at the beginning of cycle n+2 since they must wait at least one full cycle from the end of the cycle they locked their shares. All withdrawals grouped in a specific cycle may request a pro rata distributed portion of the pool's existing funds to withdraw.

This opens an attack vector where a malicious user can queue a very large amount of capital to request a withdrawal without any intention of withdrawing any amount in that cycle, diluting the other withdrawal requests by everyone else. This can then be done for every cycle, making it impossible for anyone to get their money back. To mitigate this, the Maple team makes it a pre-requirement for a withdrawal request to be carried across cycles, for the user to attempt to finalize the withdrawal for its full amount, and for it to have been partially filled if there isn't enough liquidity in the pool. If the user doesn't try to withdraw their assets, the request can only be fulfilled in round n+2, allowing other users to withdraw their funds in round n + 1.

However, under the assumption that the adversary has "unlimited" capital, it would be possible to make a large withdrawal targeting even cycles and one withdrawal targeting

odd cycles, in turn covering all rounds as queued requests can always be carried to round n+2.

Recommendation

Although the likelihood of this situation occurring is very low, it should be taken into consideration when considering potential improvements to the mechanism design that is currently implemented.

Status

Acknowledged by the team.

3S-MAPLE-06

Pool Delegate is able to block withdrawals

Id	3S-MAPLE-06
Severity	None
Difficulty	N/A
Category	Suggestion

Description

In the `WithdrawalManager.sol` contract in the `lockedLiquidity` function from contract calculates the current locked liquidity to be used in withdrawals, which cannot be used to fund open loans. It returns the distribution of available assets with the amount of shares locked for the current withdrawal window.

Users that request a withdrawal during cycle n will be able to withdraw during the withdrawal window at the beginning of cycle n+2 since they must wait at least one full cycle from the end of the cycle in which they locked their shares. Users are only able to withdraw during a withdrawal window, which starts at the beginning of each cycle.

This means that users must commit funds for up to almost three full cycles, while the pool delegate is only restricted from using these assets to fund loans during the last cycle's window, when the withdrawals mature. This can result in a scenario where the pool delegate can indefinitely lock liquidity into the protocol by funding loans with funds queued for withdrawal.

Recommendation

Although the likelihood of this situation occurring is very low, it should be taken into consideration when considering potential improvements to the mechanism design that is currently implemented.

Status

Acknowledged by the team.

3S-MAPLE-07

The require check is out of order

Id	3S-MAPLE-07
Severity	None
Difficulty	N/A
Category	Optimization

Description

In the `Liquidator.sol` contract in the function `liquidatePortion` the statements

```
uint256 returnAmount = getExpectedAmount(collateralAmount_);
require(returnAmount <= maxReturnAmount_,
"LIQ:LP:MAX_RETURN_EXCEEDED");
```

are not at the beginning of the function, which incurs unnecessary gas costs when the check inside the require is false.

Recommendation

Move these statements to the beginning of the function.

Status

Acknowledged by the team.

3S-MAPLE-08

Different modifiers for same functionality

Id	3S-MAPLE-08
Severity	None
Difficulty	N/A
Category	Suggestion

Description

The `Liquidator.sol` contract uses a `Lock` modifier while the `LoanManager.sol` contract uses a `nonReentrant` modifier to prevent reentrancy on particular functions. These implementations of the same functionality are redundant and not standardized.

Recommendation

Use a single modifier to prevent reentrant behavior across the codebase.

Status

Addressed in the following PR:

https://github.com/maple-labs/liquidations/commit/5d93d1dac375785be1ff82fc4d2ab7494ec3b8fc

3S-MAPLE-09

Function rename suggestion

Id	3S-MAPLE-09
Severity	None
Difficulty	N/A
Category	Suggestion

Description

The function `_isCollateralMaintained` in `MapleLoan.sol` contract could be called `_isCollateralAmountMaintained`. The original function name may be confusing because such naming is commonly seen on codebases to indicate whether a specific collateral is supported.

Recommendation

Rename function to `_isCollateralAmountMaintained`.

Status

Acknowledged by the team.

3S-MAPLE-10

Variable naming convention inconsistency

Id	3S-MAPLE-10
Severity	None
Difficulty	N/A
Category	Suggestion

Description

The _ suffix is used throughout the codebase whenever a variable is instantiated to memory. In the `MapleLoan.sol` contract, in the function `repossess,` the variable `nextPaymentDueDateCache's` name is inconsistent since it uses `Cache` as a suffix instead.

Recommendation

Change to the _ suffix.

Status

Acknowledged by the team, will be addressed at a later date.

3S-MAPLE-11

Redundant variable assignment

Id	3S-MAPLE-11
Severity	None
Difficulty	N/A
Category	Optimization

Description

In the `LoanManager.sol` contract in the function `triggerDefault` the assignment `liquidationComplete_ = false;` is redundant and can be safely removed.

Recommendation

Remove assignment.

Status

Addressed in the following PR:

https://github.com/maple-labs/pool-v2/commit/e8de9bf99dacad12f5804122cbb5df9cc5759c43

3S-MAPLE-12

The require check is out of order

Id	3S-MAPLE-12
Severity	None
Difficulty	N/A
Category	Optimization

Description

In the function `_disburseLiquidationFunds` in the `LoanManager.sol` contract the statements

```
address mapleTreasury_ = mapleTreasury();
require(mapleTreasury_ != address(0), "LM:DLF:ZERO_ADDRESS");
```

are not at the beginning of the function, which incurs in unnecessary gas costs when the check inside the require is false.

Recommendation

Move these statements to be the first ones in the function.

Status

Acknowledged by the team, will address at a later date.

3S-MAPLE-13

Function `_revertLoanImpairment` doesn't emit event

Id	3S-MAPLE-13
Severity	None
Difficulty	N/A
Category	Bug

Description

The function `_revertLoanImpairment` in the `LoanManager.sol` contract does not emit an update after changing the `unrealizedLosses` variable. This function is called in 2 places:

1. `removeLoanImpairment` in which the event `UnrealizedLossesUpdated` is emied after the function call;
2. `_handlePreviousPaymentAccounting` where the event is not emied.

Recommendation

Emit the event inside the internal function.

Status

Acknowledged by the team.

3S-MAPLE-14

Access to variable in storage instead of memory

Id	3S-MAPLE-14
Severity	None
Difficulty	N/A
Category	Optimization

Description

In the `MapleLoan.sol` contract in the `makePayment` function the following statement:

```
ILenderLike(_lender).claim(principal_, interest_, previousPaymentDueDate_, _nextPaymentDueDate);
```

uses `_nextPaymentDueDate` instead of `nextPaymentDueDate_` which incurs extra gas costs.

Recommendation

Change the variable used.

Status

Addressed in the following PR:

https://github.com/maple-labs/fixed-term-loan/commit/cfc507dd05c29b4c76de212e759c99a6d00c9216

3S-MAPLE-15

Unnecessary variable and require

Id	3S-MAPLE-15
Severity	None
Difficulty	N/A
Category	Suggestion

Description

In the `WithdrawalManager` contract the `processExit` function receives a `requestedShares_` amount passed as argument. This argument is required to match exactly the amount of shares locked by the account requesting the withdrawal for the current withdrawal window. This parameter is not used for anything else so it is redundant and it should be removed.

Recommendation

Remove the `requestedShares_` argument and replace it with the `lockedShares_` value in `processExit()`.

Status

Acknowledged by the team.

3S-MAPLE-16

Replace reason strings with custom errors

Id	3S-MAPLE-16
Severity	None
Difficulty	N/A
Category	Optimization

Description

To provide more information about failures, the Maple protocol implementation employs reason strings (for example, `revert("LM:RLI:NOT_AUTHORIZED")`). These are considered to be:

1. rather expensive, especially when it comes to deployment cost,
2. difficult to use with dynamic information
3. and incur extra costs when the revert condition is met.

Starting from Solidity v0.8.4, there is a convenient and gas-eicient way to explain to users why an operation failed through the use of custom errors.

Recommendation

Consider replacing all instances of the reason string require statements with custom errors across the codebase.

Status

Acknowledged by the team.

3S-MAPLE-17

Missing migration admin value check

Id	3S-MAPLE-17
Severity	None
Difficulty	N/A
Category	Optimization

Description

In the `_initialize(...)` function of `PoolManagerInitializer.sol()` contract the recipient of the new LP tokens, minted upon pool V2 deployment, is set to be the migration admin, read from storage `IMapleGlobalsLike(globals_).migrationAdmin()`.

The pool manager initialization phase of the Maple Finance V2 migration process is the responsibility of each pool delegate. Since the `_initialize(...)` function does not enforce that a migration admin must be configured priorly making it possible for a pool delegate to mint LP tokens to the `0x0` address, during poorly coordinated migration.

Recommendation

Add a `require` statement to ensure the migration manager is configured prior to pool manager initialization.

Status

Acknowledged by the team.

3S-MAPLE-18

Redundant guard condition

Id	3S-MAPLE-18
Severity	None
Difficulty	N/A
Category	Optimization

Description

In the `TransitionLoanManager.sol` contract the first condition in the following control statement of the `add(address loan_)` function is redundant and can safely be removed. This is due to the fact that `block.timestamp` should never be equal to zero.

Recommendation

Remove the extra guard.

Status

Addressed in the following PR:

https://github.com/maple-labs/pool-v2/commit/5830fa0a9f0b666bdb71ed57697e5bdb8148af27